Institute of Parallel and Distributed Systems

University of Stuttgart

Universitätsstraße 38

D-70569 Stuttgart

Master's Thesis

Speculative Reordering for a Latency-optimized Privacy Protection in Complex Event Processing

Chien-Hua, Hung

Course of Study:	Information Technology (INFOTECH)
Examiner:	Prof. Dr. Kurt Rothermel
Supervisor:	Saravana Murthy Palanisamy
Commenced:	April 23, 2018
Completed:	Octorber 23, 2018

ABSTRACT

With increasing number of applications in Internet of Things (IoT), Complex Event Processing (CEP) has already become one of the state-of-the-art technologies recently. In CEP, privacy needs to be considered carefully because events with user's sensitive information may be exposed to outside world. However, most privacy issues in CEP mainly focus on attribute-based events without considering pattern-based events. There are two important works for pattern-based privacy in CEP: suppression and re-ordering. The former suppresses events belonging to private patterns while the later tends to re-order them. The re-ordering mechanism shows better performance in terms of QoS, but the latency would be long when the size of window increases. Also, the re-ordering strategy is performed only at the end of the windows.

In this thesis, we extend the Re-ordering strategy by using speculation based on Markov chains, so we start speculating whether the private pattern occurs in current window before the end of the window. If the private pattern is predicted to occur, we then already re-order events that are part of private patterns. Additionally, the top-k preserving algorithm is introduced for preserving public patterns. Our evaluation results show that we maintain nearly 80 % utility when compared to the normal re-ordering strategy. From our experiments, it is seen that we can eliminate the time taken for re-ordering completely if the window size is greater than 3 ms.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1 Introduction

With the fast-growing number of IoT applications, IoT provides the more direct integration of the physical world by decorating our world full of networked sensors. There is a tremendous amount of IoT applications in several diverse fields such as Industrial 4.0 (automation process), Smart City (congestion control), E-health (fitness tracker), and Smart Homes (fire alarm). According to [26], there will be approximately 20.8 billion IoT devices by 2020, and an important function is to process the vast amount of raw data from the equipped sensors.

Almost all users of IoT applications are more interested in high-level meaningful information rather than raw data from sensors. Complex Event Processing (CEP) is one of the state of art paradigm that does this function that processes the stream of the raw data, and transforms it into meaningful and complex events, depending on rules provided by IoT applications. Due to the inherent distributed nature of CEP, it is an ideal candidate for many IoT applications. For instance, fraud detection is inferred when several unusual transactions of credit cards are observed and the intelligent transportation is realized by airline companies in order to track flights, track baggage, and transfer passengers. In addition, the quality of service (QoS) can also be improved while receiving the meaningful information in IoT applications.

In CEP, there are often events that are highly sensitive in terms of privacy to users, and users are unwilling to expose such privacy-sensitive events. For example, in the application of E-health, users provide a large amount of health data to their IoT devices, such as data from fitness trackers or mobile phones to their service providers, for example, insurance companies. Although they benefit from the services by sending their personal health data, they are also afraid of the consequences as privacy sensitive data might be revealed. Based on the survey in [27], more than 70% of participants are not willing to share their fitness data as their lifestyle or potential disease may be disclosed, which will lead to increasing premiums from their insurance providers. Therefore, in order to maintain users' privacy, a corresponding privacy-preserving mechanism is quite essential for users to select which types of data they are willing to share and to not share. The mechanism while preserving privacy should not deteriorate the Quality of data which in turn affects the QoS (Quality of Service). Thus QoS will deteriorate if existing complex events are not discovered (false negatives), or non-existing events are generated (false positives). In this regard two types of pattern are defined: public patterns and private patterns. The former are essential for IoT applications to provide services, and the latter are user-defined events, which should not be shared with untrusted parties. In order to protect user's privacy, several mechanisms were proposed. One straightforward approach is not to share any event of private patterns. However, some events in private patterns often belong to public patterns as well, so it turns out that many public patterns can no longer be detected, which implies that there is less QoS. In [10], the pattern-based utility suppression is proposed by suppressing parts of events in the private pattern, but it will cause low QoS if the suppressed events is part of public patterns. Instead of sharing no or any part of events, another pattern-based access control mechanism is introduced [11]. The private patterns are obfuscated by reordering events belonging to private patterns, and simultaneously the public patterns are preserved to provide high QoS.

However, most CEP applications are real-time and hence events should be forwarded as soon as possible. But in the re-ordering strategy it is necessary to wait until the least event of the private pattern is available or till a specific timeframe is lapsed in order to know whether a private pattern had occurred or not and then reorder. This would contradict a real time system. In other words, in this thesis we extend the re-ordering strategy provided by [11] and develop a speculative re-ordering strategy based on Markov-chains, which speculate private patterns even before it completes and also reorder thus reducing the time taken for re-ordering after private pattern has completed or the timeframe window for that private pattern has expired. Therefore, the main goal of this Speculative Reordering strategy is to minimize the time taken by the non-speculative re-ordering strategy after the window completion while still maintaining the QoS guarantees.

Chapter 2 Related Work and Problem Statement

2.1 Relate Work

As mentioned in the introductory chapter, the main purpose of Complex Event Processing (CEP) [9] is to deal with temporal relationship between events such as sequence matching among event streams. Thus CEP has been one of the emerging technology in which both researchers [1], [2], [3] as well as industries [4], [5] have invested time and effort in developing an efficient CEP system. In CEP, privacy patterns are inferred by observing relationships between incoming events, so user's sensitive patterns may also be discovered while dealing with events that are part of patterns related to user's behavior. Therefore, privacy should be taken into consideration carefully in CEP system because users are not willing to expose their privacy to untrusted parties. However, most privacy issues in CEP have focused on individual events rather than pattern-based events. For example, there are some approaches about privacy in CEP, such as differential privacy in private data stream [22] , zero-knowledge privacy guarantees [23], all of which are at the level of attribute-based events. But often privacy is revealed in terms of pattern. Thus, in this thesis, we focus on pattern-based events which contain both public patterns and private patterns of the input data stream to CEP systems.

In [10], the authors proposed a pattern-based approach in which private patterns are concealed by suppressing events belonging to private patterns. Nevertheless, suppressed events might also be part of public patterns, so the loss of public patterns persists in this approach, which may result in low QoS. Therefore, in order to maintain user's privacy, QoS should not be affected when concealing events that are part of private patterns. Instead of suppression, another pattern-based access control mechanism is proposed in [11] by re-ordering events that belong to private patterns. In this way, public patterns can be preserved, and high QoS is guaranteed simultaneously. Clearly, the re-ordering mechanism has the better performance in terms of QoS than suppression approach when maintaining pattern-based privacy.

However, the re-ordering approach may take a longer processing time when the window size becomes longer or when the number of patterns goes high. Also, re-ordering to conceal private patterns is done only at the end of window because all events need to be observed. In order to improve latency, in this thesis, we extend the existing re-ordering strategy to a Speculative Re-ordering strategy where completion of private patterns are predicted well before the end of the window and also reordered. There are several research on speculating event patterns. For instance, in [13], speculation is regarded as a classification problem that can be solved by using Singular Value Decomposition (SVD) and Support Vector Machine (SVM) model. In [14], a generic algorithm is implemented to learn predictive patterns from sequence events as early as possible [14]. A framework with association rule mining is proposed in [15], and the goal is to detect target patterns by recognizing events that occur frequently before target patterns happen. Unfortunately, those approaches have a limitation: Their target patterns are usually composed of rare events such as equipment failure or anomaly detection, which do not consider the case of frequent patterns.

For frequent patterns, a speculative model is presented in [16] by deploying decision trees and Piecewise-Constant Conditional Intensity Models. The model proposed in [16] is trained from dependencies among sequences of incoming events, and it can predict the occurrence of target patterns in a given time interval. In addition, the event forecasting with Markov-chains model is introduced in [17], [18]. The approaches in [17], [18] are able to estimate when the target pattern is expected to be matched, and they focus on the completion time of target patterns. Nonetheless, these mechanisms only take continuous events into consideration, and lack the ability to deal with discrete events. In this thesis, we basically focus on the data stream containing discrete-time events, and we view our target pattern as private patterns using Markov-chain predictions.

Additionally, it is not enough to take only the private pattern into consideration while re-ordering events, and it is also difficult to know the number of public patterns, which are involved into re-ordering process. The top-k strategy of selection algorithm has been discussed in a huge amount of studies. In [20], the authors introduced the Greedy algorithm for mining top-k influential nodes in order to maximize the spread of the influence (further adoptions of the new product). Numerous top-k processing techniques are classified in [21], which shows that top-k queries are effective while dealing with the massive amount of data in domains such as Web, multimedia and distributed systems. In [19], the SPECTRE system with speculative approach shows a good scalability by processing k most promising window versions. In short, the top-k selection algorithm is useful when dealing with a huge amount of data and aiming to keep a good scalability at the same time. Thus, in this thesis we use the top-k strategy to preserve the top-k utility maximizing public patterns along with the speculative strategy.

2.2 Problem Statement

In pattern matching of CEP, the goal is to make sure that QoS is maintaining high while maintaining user's privacy by re-ordering events that are part of private patterns. However, QoS is affected by false positive events and false negative events, which are maybe introduced after the re-ordering process. Therefore utility is calculated in the non-speculative reordering strategy as follows: [11]

$$\begin{aligned}\text{Utility (U)} = {} & \Sigma_{i=1}^{\#\, of\ matched\ public\ patterns} w_i \\ & - 2 * \Sigma_{j=1}^{\#\, of\ matched\ false\ positives} w_j \\ & - \Sigma_{k=1}^{\#\, of\ matched\ private\ patterns} w_k \end{aligned} \qquad (2\text{-}1)$$

where w_i, w_j are the user-defined weight of public pattern onto the QoS, and w_k is the weight of private patterns defined as below, and we use the same utility function to evaluate our QoS :

$$w_k = (\Sigma\, w_i + 1) * cp_k \qquad (2\text{-}2)$$

where cp_k is the tuning factor for trade-off between privacy and QoS.

The problem now is to find a proper speculative re-ordering strategy that predicts whether the private pattern occurs in current window via Markov-chains modeling. Thus, the overall processing time can be improved, which leads to shorter latency, while maintaining. More precisely, the utility of speculative re-ordering method should approach the utility of non-speculative reordering strategy as closely as possible.

Chapter 3 Basic Concept

In this chapter, two main concepts are introduced. First, we introduce the concept of Complex Event Processing (CEP). Secondly, the concept of Markov-chain model is presented.

3.1 Complex Event Processing

In this section, we introduce CEP systems, useful operators, and privacy issues in CEP systems. Complex event processing (CEP) is the process which aims to filter, combine and interpret a series of input data events in order to infer high-level information based on a set of user-defined rules and patterns. There are diverse applications of CEP, such as financial analysis in stock market or traffic monitoring for traffic jams or accidents. Among above applications, CEP is responsible for processing, analyzing and correlating the input data stream in order to obtain more complicated information from different sources. In other words, CEP provides solutions to cope with real-time data from a great deal of sources such as IoT sensors.

3.1.1 Complex Event Processing System

In 1997, Rosenblum and Wolf [24] proposed the event processing engine with a publish-subscribe feature, which is regarded as the first prototype of CEP engines. As shown in Figure 3-1, the conventional CEP system is presented with event observers, consumers and the CEP engine in the middle. The event observers are responsible for capturing events, which happened outside the systems, and publishing the notification of events to CEP engine. The CEP engine then filters, aggregates or combines those notifications in order to derive more complicated events, also known as complex events. At the end, the event consumers subscribe complex events from the CEP engines, and discover the more high-level information in which they are interested. Therefore, through CEP systems, information sources publish the notification of events to CEP engines, and users can subscribe a series of user-concerned events at the same time. In this process, the

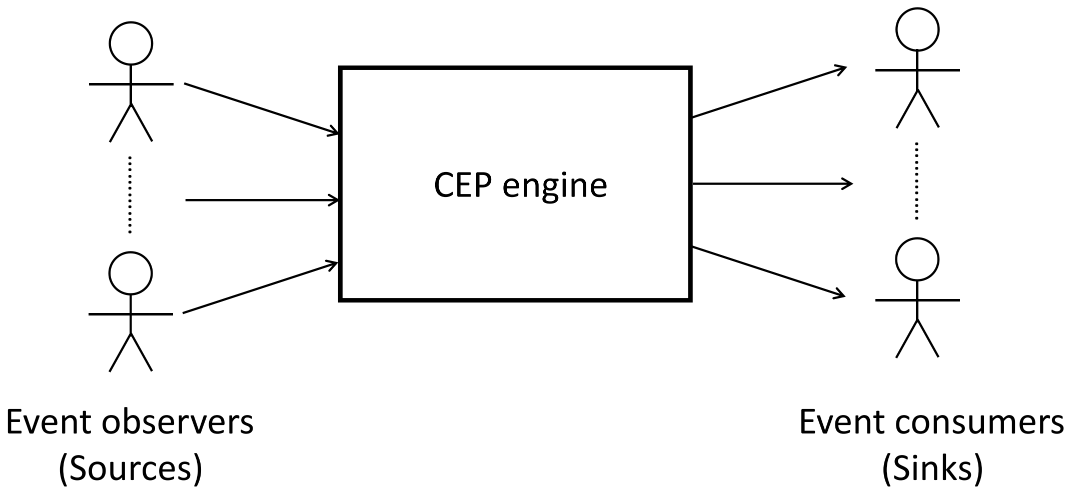


Figure 3-1 High view of Complex event systems

CEP engine acts as the middleware between information sources and sinks to deal with event notifications for the purpose of generating useful complex events. There are two types of CEP systems: topic and content-based systems. In topic-based systems, event consumers can subscribe concerned events, and event observers would pick up topics related to consumers' interest before publishing. On the other hand, in content-based systems, event observers publish all events, and by using complex event filters consumers can choose events, which contain the content based on their pre-defined rules.

3.1.2 Basic Model Operators

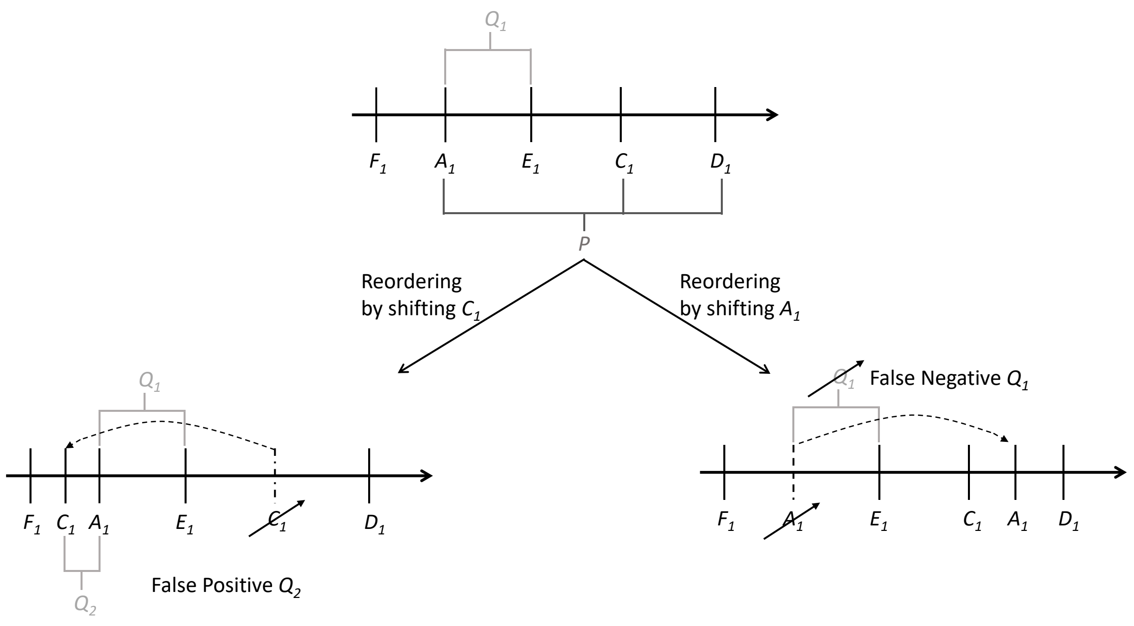
As mentioned in [25], there is a majority of IFP models, such as function model, processing model, and language model, each of which has the different purpose. In this section, we mainly concentrate on operators in language model because they are more related to the implementation of CEP in this thesis. The language model is more precise and detailed description of given rules with many operators. It should firstly focus on specific classes, and suitable operators are chosen for specialized classes. For instance, in case of logic operators, a conjunction is defined as the situation where all given items have been detected, while a disjunction is satisfied when at least one of the given items has been detected.

For pattern-based applications in CEP systems, the sequence operators are the most important operator for pattern-based events. Since the pattern consists a set of ordered items, and sequence operators not only take responsibility for observing the arrival of a set of items but also consider the order of arrival time of items. To be more precise, a sequence is defined as a set of information items with specified order, and it is matched only when the sequence operator detects all items in a pre-defined order. In most cases, the ordering relationship is based on timestamps of incoming data flow in CEP systems.

Another useful operator is the window operator in pattern-based event processing. Due to the unbounded assumption in CEP systems, most language models do not handle all input data flow at one time, but deal with small chunks of incoming data repeatedly. The window operator is defined as a range, which only contains a portion of input data flow. Every time when CEP systems receive the new incoming window, which contains finite input data stream via window operators, systems can easily cope with the bounded data. The major types of window operators are time-based (logical) and content-based (physical). The former treats bounds as a function of time while the latter views bounds as the number of items. For example, the bound can be regarded as items in five minutes for time-based type, or first five items for content-based type. In this thesis, we assume windows to be non-overlapping for simplicity, so windows are considered as disjoint windows, which are most common type in many applications. We also have fixed size windows with equally shifting lower and upper bound, which makes it a proper and powerful tool for continuously arriving data flow.

3.1.3 Privacy in Complex Event Processing

In CEP, some complex events may contain patterns with sensitive information of users. By observing those complex events, an authorized third party could infer the behavior of users. Patterns with sensitive information about users are called private patterns, which should not be exposed to untrusted parties. On the other hand, the public pattern is defined as the essential pattern, which is required for the CEP application. During the process of detecting complex events, false negatives will occur if existing complex events are not discovered, and QoS may be degenerated. Furthermore, non-existing events might sometimes accidentally be generated, which leads to false positives.



Q_1
F_1 A_1 E_1 C_1 D_1
P
Reordering by shifting C_1
Reordering by shifting A_1
Q_1
F_1 C_1 A_1 E_1 C_1 D_1
Q_2
False Positive Q_2
Q_1 False Negative Q_1
F_1 A_1 E_1 C_1 A_1 D_1

Figure 3-2. Reordering introducing False Positive and False Negative

For example, in order to conceal the private pattern, the re-ordering strategy is demonstrated as shown in Figure 3-2. In this example, the private pattern (P_1) and public patterns (Q_1, Q_2, Q_3) are defined as following:

$$Q_1 = \{A, E\}\ , Q_2 = \{C, A\},\ Q_3 = \{C, D, F\}$$

$$P_1 = \{A, C, D\}$$

By reordering the pair $\{A_1,C_1\}$ in P_1, the private pattern can be concealed. However, in case of shifting C_1 before A_1, the false positive of public pattern Q_2 (C_1, A_1) is introduced into the modified event stream. On the other hand, the public pattern *Q1* (A_1, E_1) is no longer presented in the modified event stream while shifting A_1 after C_1, which leads to a false negative.

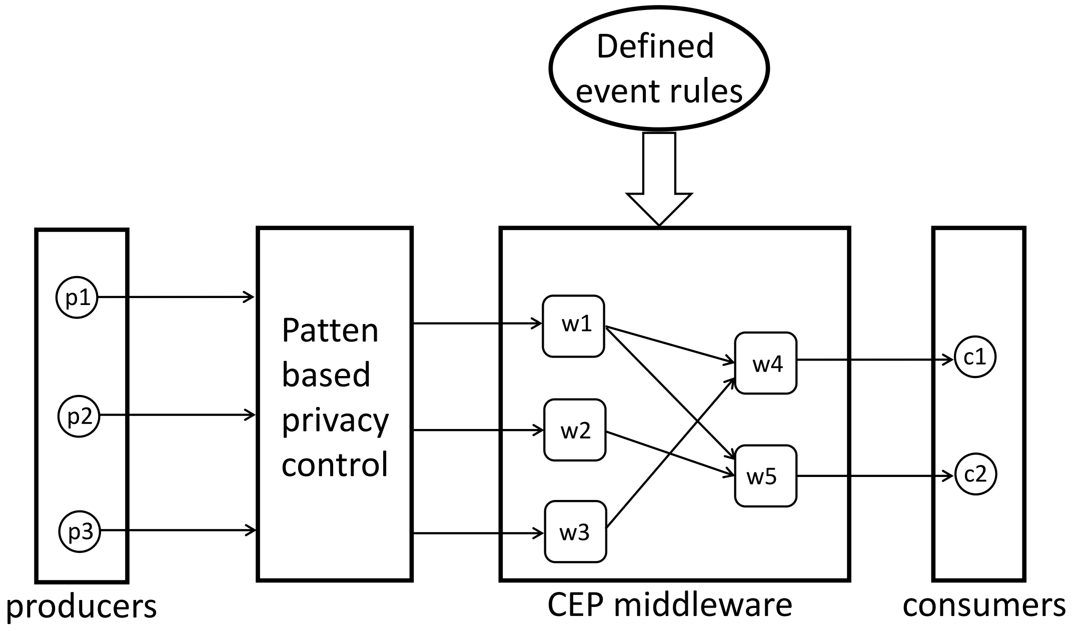


Figure 3-3 Simplified CEP system model

The overall CEP system model is shown in Figure 3-3. There are producers and consumers on the left and right side. In real-time applications, producers can be any type of sensors controlled by users, and consumers could be IoT service providers. Additionally, the CEP middleware may be the extension of IoT service and mainly contain sequence and window operators with user-defined event rules. In our assumption, producers are trusted while CEP consumers are untrusted parties, who try to acquire sensitive information by manipulating CEP middleware to observe private patterns from producers. Suppose the fitness service as an example. Users are able to send their fitness data to cloud service (CEP middleware) via IoT sensors (producers) and receive the feedback about their recent fitness behavior (private patterns), such as running time or heartbeat during exercise. However, IoT service providers (consumers) are able to analyze fitness data from users, and sell the result to outside companies. For instance, the insurance company can modify the insurance fee after receiving the fitness data of customers from IoT service providers. Therefore, a pattern-based privacy control system is introduced before sending raw data to CEP middleware, and its goal is to conceal user’s private pattern by re-ordering events that belongs to private patterns.

3.2 Markov-chain Model

In this section, there are several basic concepts such as fundamental definitions, transition matrix, and the characteristic of transition steps. In modern probability, the processes between different moves assumes that the previous outcomes may be related to the future outcomes, which implies that after observing a set of outcomes of an experiment, the past outcomes could affect the prediction of outcomes in future experiments. In 1907, A. A. Markov started the new assumption of chance process. His idea is to simplify the process which only takes present outcomes into consideration to predict the future outcomes of the next experiment. The mathematical system with this assumption is called the Markov chain.

3.2.1 Definition of Markov Chain

Assumed that there is a set of states, $S = \{X_0, X_1, X_2, \ldots\}$. We define this set of states as the process of Markov chain, and a few definitions are shown below.

Definition 1: The state space of a Markov chain S is the set of values that each X_t can take, where X_t represents the value of a state in the Markov chain process at time t. For example, if $X_2 = 3$, we say the process at time 2 is 3.

Definition 2: A trajectory (i.e. the path in Markov chain) of a Markov chain is a particular subset of Markov chain process S with values for $X_0, X_1, X_2, \ldots$.
For example, if $S = \{1, 2, 3, 4, 5, 6, 7\}$, the trajectory up to time t = 4 is 1, 2, 3, 4, 5. Thus, more specifically, the trajectory up to time i means that $X_0 = s_0, X_1 = s_1, X_2 = s_2, \ldots, X_i = s_i$.

Moreover, the fundamental property of the Markov chain is described as follows: only the most recent point in the trajectory affects what happens next. In other words, the state X_{t+1} only depends on the previous state X_t, and other previous states such as X_{t-1}, X_{t-}

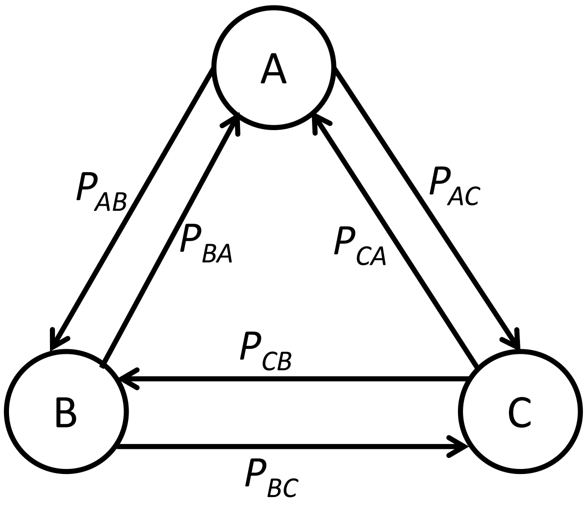


Figure 3-4. Transition diagram with states {A, B, C}

$_{2,}$ … X_0 have no influence on the present state X_{t+1}. According to this property, we have the final definition as below.

Definition 3: For s set of states $S = \{X_0, X_1, X_2, \dots\}$, it is the process of the Markov chain if it satisfies Markov chain property as below.

$$\mathrm{P}(X_t = \mathrm{s} \mid X_{t-1} = s_{t-1}, \dots, X_0 = s_0) = \mathrm{P}(X_t = \mathrm{s} \mid X_{t-1} = s_{t-1})\ ; \quad (3\text{-}1)$$

$$\text{where for all } \mathrm{t} = 1, 2, 3, \dots \mathrm{t}\text{ , and for all states } s_0, s_1, \dots s_t$$

3.2.2 Transition Matrix

For the purpose of better understanding of transition between different states in Markov-chain process, the transition diagram of Markov chains is introduced. For example, the transition diagram of Markov chain with three states is shown in Figure 3-4. In Figure 3-4, nodes represent each state and arrows correspond to the path of transition probability between different states. Based on the transition diagram in Figure 3-4, the corresponding matrix which states the Markov chain can be described as below:

$$\mathrm{P} = \begin{bmatrix} P_{AA} & P_{AB} & P_{AC} \\ P_{BA} & P_{BB} & P_{BC} \\ P_{CA} & P_{CB} & P_{CC} \end{bmatrix} \qquad (3\text{-}2)$$

There are several properties in the transition matrix. First, the rows represent the current state (i.e. from X_t), and the columns represent the next state (i.e. to X_{t+1}). Second, the entry (i, j) in the matrix is the conditional probability which shows transition from state i to state j. p_{ij} is denoted as $\mathrm{P}(X_{t+1} = \mathrm{j} \mid X_t = \mathrm{i})$ for i, j $\in \{A, B, C\}$, and t = 0, 1, 2, ... By doing so, the transition matrix of Markov chain can be defined as $\mathrm{P} = \left(p_{ij}\right)$. The Last but not least, the sum of each row in the transition matrix must be equal to one but the sum of each column does not generally need to be one.

3.2.3 Transition after t-step

As mentioned before, the probability between different states is only dependent upon the current state. Assume $\{X_1, X_2, X_{3,} \dots\}$ denotes the Markov chain with state space $S = \{1, 2, 3, \dots, n\}$, and the elements in the transition matrix P can be derived as following :

$$(\mathrm{P})_{ij} = p_{ij} = P(X_1 = j \mid X_0 = i) = P(X_{n+1} = j \mid X_n = i) \ \forall\, n \in S; \qquad (3\text{-}3)$$

where p_{ij} is the probability of making a transition from state i to state j in a single step.

We can extend the above equation of single step to the t-step transition probabilities given by the matrix P^t for any t as below :

$$(P^t)_{ij} = P(X_t = j \mid X_0 = i) = P(X_{n+t} = j \mid X_n = i) \ \forall n \in S \qquad (3\text{-}4)$$

The theorem of t-step transition has been already proved, so we do not delve deeper into it.

Chapter 4 Proposed Model

In this chapter, we introduce the proposed model with speculative re-ordering algorithm. There are four main parts in the proposed model. They are speculation, top-k preserving, re-ordering and utility comparison. First, the overview of the proposed model is described. Also, the fundamental aspect of discrete Markov-chain model is introduced for pattern-based speculation. The discrete Markov-chain model is implemented as an offline learning model to obtain speculative parameters. Before re-ordering, the most important patterns are preserved by top-k preserving strategy. Moreover, the graph-based algorithm is applied for re-ordering. At the end, the utility between speculative re-ordering and non-speculative re-ordering algorithm is compared for evaluation. For each model, we give a detailed explanation and description with flow graphs explaining the procedure.

4.1 Overview

As shown in Figure 4-1, the whole overview of the proposed model is described along with these function blocks. In this thesis, the goal is to maintain the pattern-based privacy of data stream with given public patterns and private pattern. First of all, assume that there is the input data stream for processing with specific fixed window size. Each window of data stream is sent to speculative trained model. The model is trained by Markov chains with data that has the same characteristic as input data stream. For example, we can train the health data of patient last year to build the model for future speculation. After training, we obtain the transition matrix based on Markov-chain model, prediction point, and threshold. With above output from trained model, we can speculate whether the private pattern occurs in each incoming window. If the transition probability derived from speculation is greater than threshold, the window will be labeled as positive, which indicates that there is the private pattern in this window. Otherwise, the window will be labeled as negative.

For positive windows, we take only the part up to prediction point into re-ordering. Before re-ordering, top-k preserving strategy is implemented to preserve the most important k public patterns in each window. After top-k preserving, the re-ordered pairs derived from private pattern are sorted based on estimation weight. In re-ordering, the

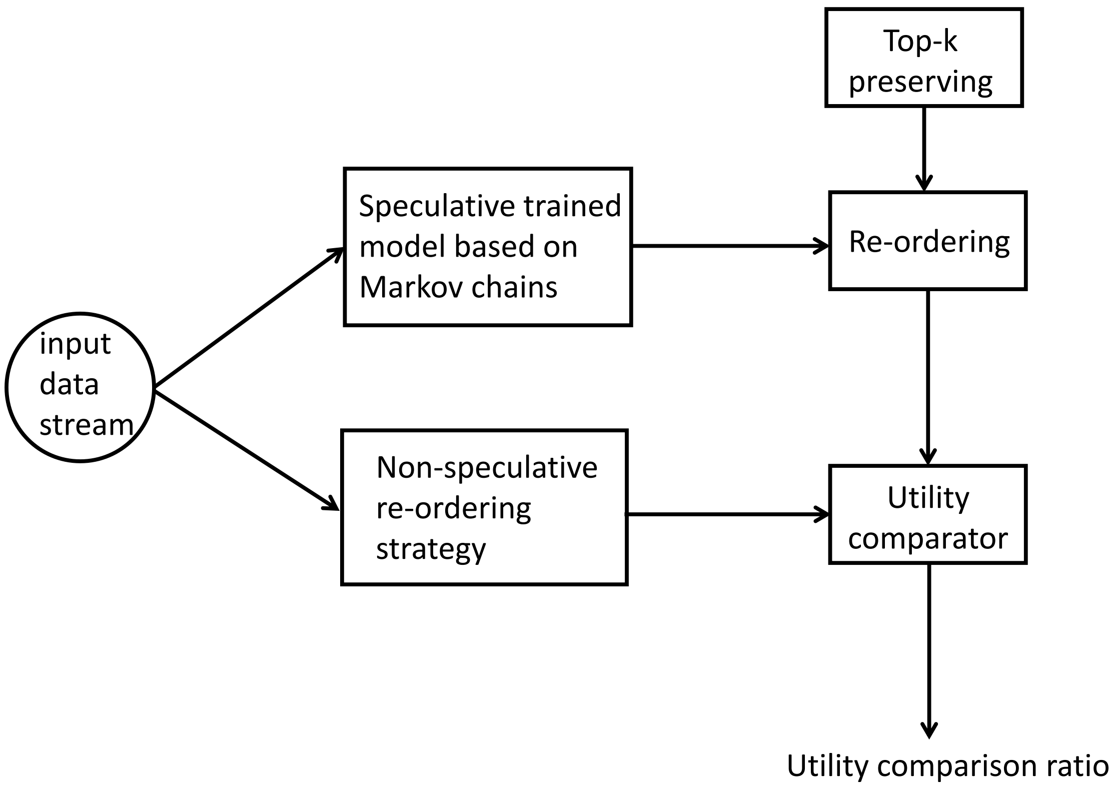


Figure 4-1 Overview of proposed model

graph-based re-ordering mechanism is implemented from [11], and the re-ordered window is obtained. At the end, the utility of speculatively re-ordered window is compared to the utility of non-speculatively re-ordered window.

4.2 Discrete Markov-chain Model

In Complex Event Processing (CEP), a pattern is matched only when events of pattern occur in correct order. For instance, a pattern is defined as P_1 = {A, C, D}. When event ‘A’, ‘C’, ‘D’ are sequentially observed, pattern P_1 is matched. Therefore, the relationship between events of defined pattern must be taken into consideration when speculating whether the pattern occurs in the current window. In other words, if the relationship between previous event and current event is traceable, the occurrence of pattern will be predictable.

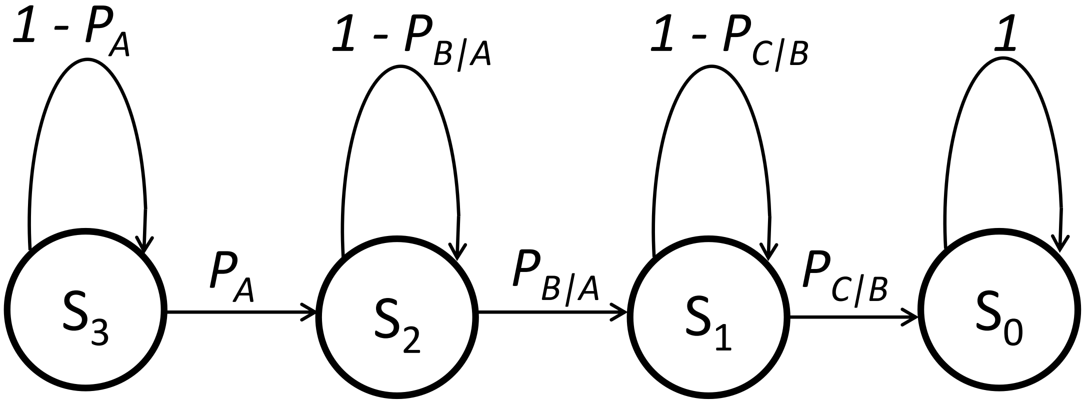


Figure 4-2. State diagram of pattern P = {‘A’, ‘B’, ‘C’}

As discussed in previous chapter, Markov-chain model assumes that future outcomes only depend on the current state. With the conditional probability of current state given previous state, the occurrence of next state can be predicted. Thus, in pattern speculation, different states represent different event relationships, and the pattern can be speculated by implementing Markov-chain model. Since our events in data stream are not continuous, we use discrete Markov chains to forecast the occurrence of patterns. In order to describe more precisely how discrete Markov chains implement into pattern speculation, we give an example below.

Suppose that event types *E* contains ‘a’ to ‘z’ and ‘A’ to ‘Z’, and the pattern *P* equals to {*‘A’, ‘B’, ‘C’*}. In Markov-chain model, each state represents the specific outcome. For pattern speculation, each state is defined as the number of remaining events of pattern. For example, S_k is denoted as the state, which still needs to receive k remaining events of pattern before the pattern is totally matched. The state diagram of pattern *P* is shown in Figure 4-2.

In Figure 4-2, there is initially no observing event of pattern *P*, so the initial state is denoted as S_3, which indicates that there are three remaining events ‘A’, ‘B’, ‘C’. The state stays in S_3 until the event ‘A’ comes. When the new coming event is ‘A’, the state shifts to S_2 with the probability P_A. In S_2, the state keeps staying in S_2 with the probability $1 - P_{B|A}$, where $P_{B|A}$ is the conditional probability of event *B* given event *A*. At the end, in final state S_0, all events belonging to pattern *P* are all observed in correct order, so the pattern *P* is entirely matched. Moreover, no matter what event comes, the state stays in S_0 until all coming events are detected.

In order to derive the transition matrix of state diagram in Figure 4-2, we consider the coming window *w* and the state matrix *R* shown respectively as following:

w = {‘e’, ‘y’, ‘H’, ‘a’, ‘k’, **‘A’**, ‘D’, ‘b’, ‘g’, ‘E’, ‘F’, ‘G’, ‘H’, **‘B’**, ‘Y’, ‘d’, ‘m’, ‘p’, ‘o’, ‘n’, **‘C’**, ‘K’, ‘z’}; (4-1)

R is initialized as $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$ (4-2)

Before observing event ‘A’, the state keeps in S_3, and there are five events before event ‘A’ is observed. Thus, the number of events in S_3 is five, and *R* becomes as below

$$R = \begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4-3}$$

After detecting event ‘A’, the state shifts to S_2, and *R* is shown as following:

$$R = \begin{bmatrix} 5 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4-4}$$

Similarly, after detecting event ‘B’, there are seven events in S_2 , and equation (4-4) becomes as below:

$$R = \begin{bmatrix} 5 & 1 & 0 & 0 \\ 0 & 7 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4-5}$$

At the end, when observing all events including event ‘C’ in window *w*, the final *R* can be derived as following:

$$R = \begin{bmatrix} 5 & 1 & 0 & 0 \\ 0 & 7 & 1 & 0 \\ 0 & 0 & 6 & 1 \\ 0 & 0 & 0 & 2 \end{bmatrix} \tag{4-6}$$

Hence, if each element in equation (4-6) is divided by the sum of its corresponding row, we can derive the transition matrix T of Markov-chains model with given pattern P = {‘A’, ‘B’, ‘C’} for incoming window w as below:

$$T = \begin{bmatrix} \frac{5}{6} & \frac{1}{6} & 0 & 0 \\ 0 & \frac{7}{8} & \frac{1}{8} & 0 \\ 0 & 0 & \frac{6}{7} & \frac{1}{7} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4-7}$$

In each row of transition matrix T, the first non-zero element is the probability that stays in the current state, and the second non-zero element represents the conditional probability of the next state given specific relationship of events. Take the second row for instance. In the second row, the probability that shifts from S_2 to S_1 is 1/8 when detecting the event ‘B’, and the probability that stays in current state is 7/8. Note that the sum of each row should be equal to one due to the property of Markov-chain model.

With transition matrix, we can speculate whether the pattern occurs after observing several incoming events based on the theorem of t-steps transition in Markov-chain model. The transition matrix after t steps is the t-times polynomial of transition matrix itself. For instance, assume that there will be ten events coming. The transition matrix after observing ten events T^{10} are derived as below:

$$T^{10} = \begin{bmatrix} 0.16150558 & 0.40627997 & 0.26887591 & 0.16333854 \\ 0.00000000 & 0.26307558 & 0.34312082 & 0.39380360 \\ 0.00000000 & 0.00000000 & 0.21405832 & 0.78594168 \\ 0.00000000 & 0.00000000 & 0.00000000 & 1.00000000 \end{bmatrix} \tag{4-8}$$

In each row of T^{10}, the last element represents the occurrence probability of pattern {‘A’, ‘B’, ‘C’} after observing ten events given specific events. Take the third row in T^{10} as an example, if event 'A' and 'B' are already detected, the probability that pattern *P* occurs after next ten events is 0.78594168. Furthermore, when the number of next coming events increases dramatically, the occurrence probability of pattern *P* in each row will approach one very closely. Take sixty events for example, and the transition matrix T^{60} after observing next sixty events is shown as following:

$$T^{60} = \begin{bmatrix} 0.00000000 & 0.00125501 & 0.00494069 & 0.99378655 \\ 0.00000000 & 0.00033150 & 0.00164707 & 0.99802143 \\ 0.00000000 & 0.00000000 & 0.00000962 & 0.99990379 \\ 0.00000000 & 0.00000000 & 0.00000000 & 1.00000000 \end{bmatrix} \quad (4\text{-}9)$$

From T^{60}, we can conclude that after observing sixty events, the probability that detects the pattern {'A', 'B', 'C'} is very high because the occurrence probability of pattern *P* in every state is nearly one.

4.3 Offline Learning

In offline learning, the goal is to obtain the transition matrix, prediction point and threshold by implementing Markov-chain model in order to forecast whether the pattern occurs in every coming window. As discussed previously, the transition probability in transition matrix is denoted as the occurrence probability of defined pattern in specific state. In each coming window, if the transition probability is greater than the user-defined threshold after observing several events, i.e. several steps transition, the pattern will be highly expected to occur in current window. However, if the transition probability is less than the threshold, the pattern will not occur in current window.

Figure 4-3 shows the overview of the offline learning model in order to derive the transition matrix, proper prediction point and threshold. First of all, there are two phases for generation of transition matrix: training phase and testing phase. In training phase, the transition matrix is trained with a training dataset. After training, the testing dataset is generated with the same characteristic as training dataset. The transition matrix is evaluated in testing phase by precision and recall rate. Finally, based on the result of recall and precision, a proper threshold and its corresponding prediction point are determined.

At the beginning of offline learning, there is an event generator, which can randomly generate the dataset based on given event types from A - Z and a - z.

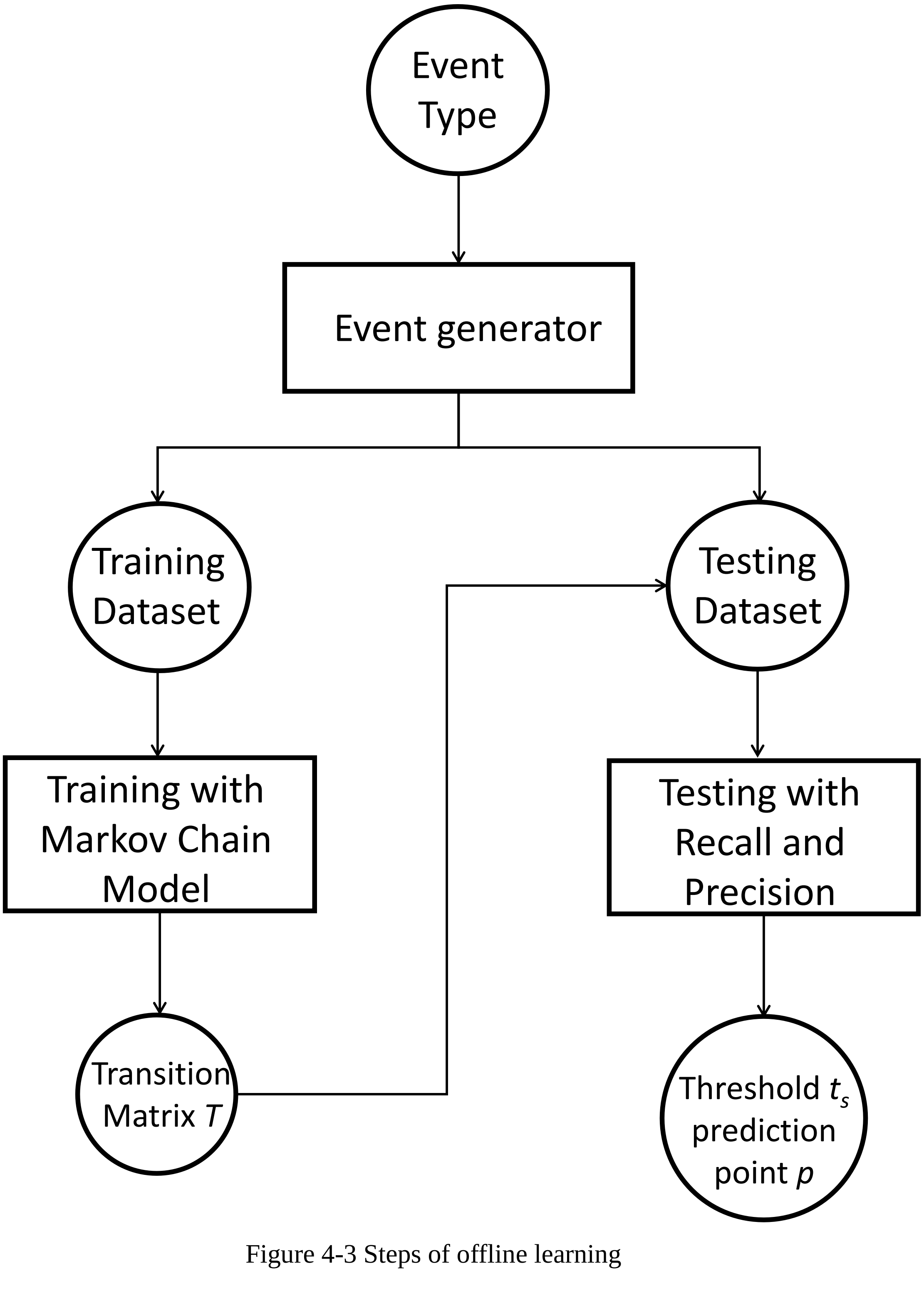

Event Type
Event generator
Training Dataset
Testing Dataset
Training with Markov Chain Model
Testing with Recall and Precision
Transition Matrix T
Threshold t_s prediction point p

Figure 4-3 Steps of offline learning

For training dataset, the occurrence probability of user-defined pattern is chosen to be around 0.5, so that the results can be clearly seen, which is not the case for rare private patterns.

The relationship between the user-defined tolerance ε, which is set to 0.5, and occurrence probability p_o is defined as below:

$$|p_o - 0.5| \le \varepsilon \qquad (4\text{-}10)$$

We generate the training dataset in such a way that the inequality (4-10) is satisfied. Now the transition matrix T is derived by implementing the Markov-chain model. The transition matrix T is able to predict the transition probability of given pattern if the processing dataset has the same characteristic as the training dataset. In addition, the number of windows in training dataset should be large enough to guarantee that the transition matrix is stable and reliable after training.

The difference between training and testing is that the training dataset is implemented to build up a trained model while the test dataset is to validate the built model. Moreover, the size ratio between training dataset and test dataset is chosen as 8 : 2. In testing phase, the transition matrix T is validated by applying the test dataset, and the purpose is to learn the proper threshold and corresponding prediction point. The range of threshold starts from 0.5 to 0.95, and the corresponding prediction point of given threshold is obtained when the transition probability after observing several events is greater than given threshold.

In order to decide proper threshold and prediction point, the recall and precision rate are introduced. As shown in Figure 4-4, false positives are denoted as items, which are incorrectly labeled as belonging to occurred patterns after prediction. Similarly, false negatives represent items, which are not labeled as belonging to occurred patterns but they should be. The formulas of recall and precision are described as following:

$$\text{Precision} = \frac{True\ Positives}{True\ Postives + False\ Positives} \qquad (4\text{-}11)$$

$$\text{Recall} = \frac{True\ Positives}{True\ Postives + False\ Negatives} \qquad (4\text{-}12)$$

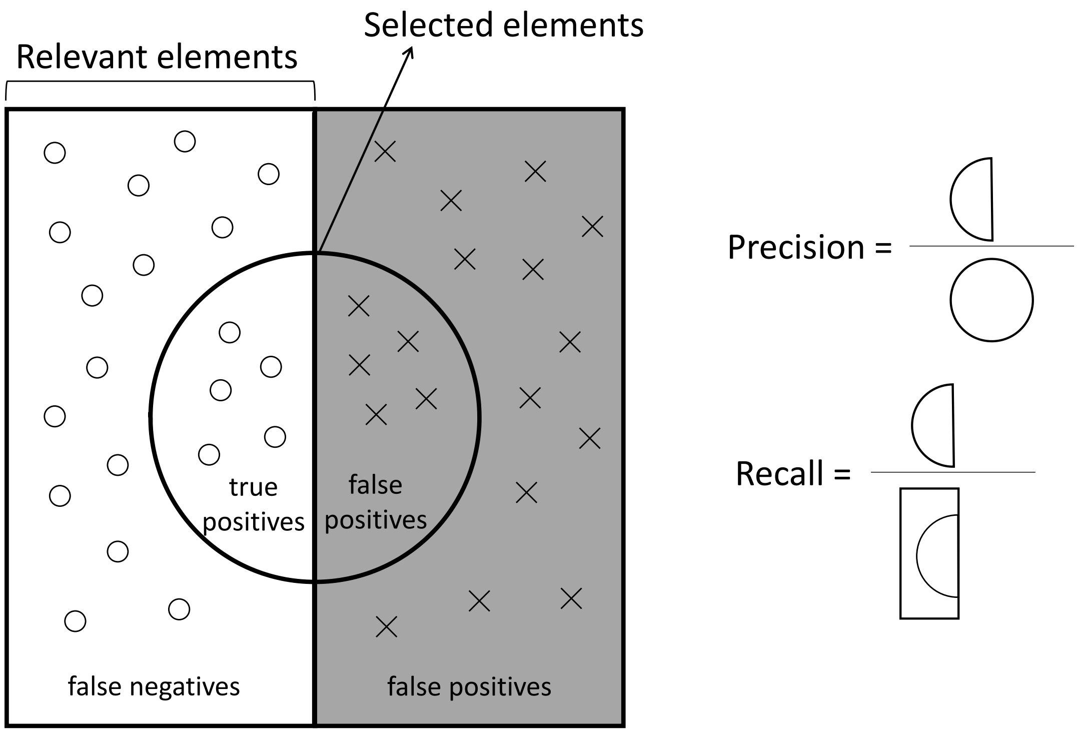
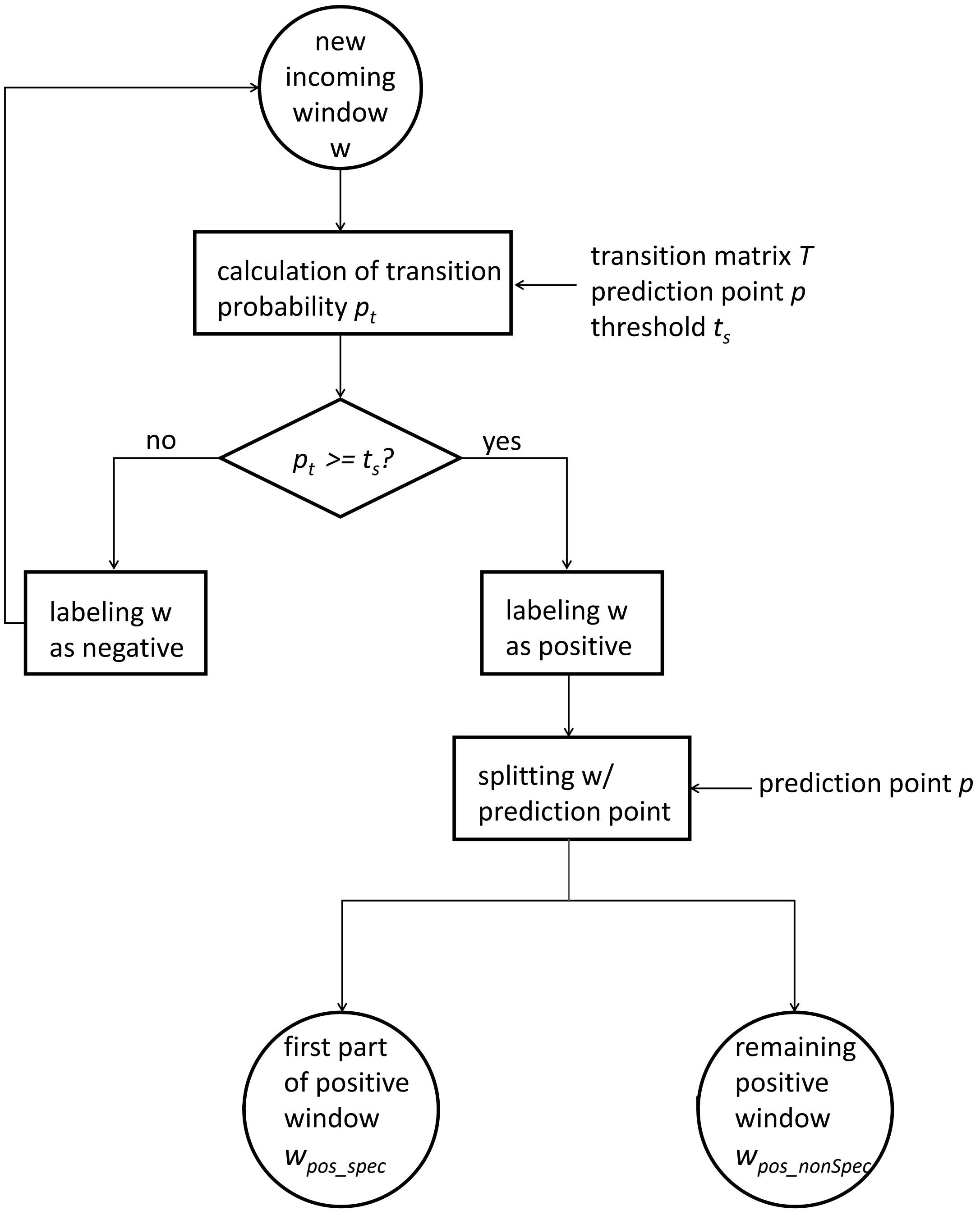


Figure 4-4. Diagram of recall and precision

From equation (4-11) and (4-12), recall can be interpreted as the rate of relevant items that are selected, and the precision is regarded as the rate of selected items that are relevant. Hence, based on results of recall and precision, an optimal threshold and its corresponding prediction point are determined that has a relatively high recall and precision.

4.4 **Speculation**

With threshold, prediction point and transition matrix derived from offline learning, we can speculate whether there is a private pattern in window of new incoming dataset. The overview of speculation phase is depicted in Figure 4-5. When dealing with new incoming data stream, the input data stream is first divided into many windows with same size. For each window, all events before prediction point *p* are observed, and we can learn the relationship between events belonging to private pattern before prediction point. Based on observation of those events, the transition probability is calculated with transition matrix *T*. If the transition probability at prediction point *p* is greater than threshold t_s, the current processing window will be labeled as positive, which implies that the private pattern exists in this window. On the contrary, if the transition probability is



new incoming window w
calculation of transition probability p_t
transition matrix T
prediction point p
threshold t_s
$p_t >= t_s$?
no
yes
labeling w as negative
labeling w as positive
splitting w/ prediction point
prediction point p
first part of positive window w_{pos_spec}
remaining positive window $w_{pos\_nonSpec}$

Figure 4-5 Procedure of speculation phase

less than the threshold, the private pattern is not happening in this window, and it will be labeled as negative. The speculative algorithm based on Markov-chain model is described in Algorithm 4.1.

Algorithm 4.1　Speculation based on Markov-chain model

```
procedure MARKOV-CHAINSPECULATION (predictionPoint $p$, threshold $t_s$)
    for each window in dataStream do
        $prob \leftarrow window$.getTransitionProb($p$)
        if $prob >= t_s$ then
            window.label(positive)
            $w_{pos\_spec} \leftarrow window$.first($p$)
            $w_{pos\_nonSpec} \leftarrow window$.remain($p$)
        else
            window.label(negative)
        end if
    end for
end procedure
```

For example, assuming prediction point p = 40, threshold t_s = 0.8, pattern P = {'A', 'B', 'C'}, window size = 100, and the transition matrix is defined as below:

$$T = \begin{bmatrix} 0.966 & 0.034 & 0 & 0 \\ 0 & 0.959 & 0.041 & 0 \\ 0 & 0 & 0.961 & 0.039 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4-13}$$

Since the prediction point p is 40, there are still 60 events left behind, which indicates that the transition steps are 60. Thus, when receiving the new coming window, the transition matrix at the prediction point p is derived as following:

$$T^{60} = \begin{bmatrix} 0.126 & 0.216 & 0.262 & 0.396 \\ 0 & 0.081 & 0.221 & 0.698 \\ 0 & 0 & 0.092 & 0.908 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4-14}$$

Based on transition matrix T^{60}, transition probabilities of different states are obtained.

For example, in the third row of T^{60}, the transition probability 0.908 is greater than threshold t_s 0.8, which indicates that the pattern {'A', 'B', 'C'} may occur in state S_1, which has already observed event 'A' and 'B'. In other words, if there exist event 'A' and 'B' before prediction point *p,* i.e. in first 40 events, there may be the event 'C' coming in the remaining 60 events, so the processing window will be labeled as positive.

At the end, windows labeled as positive are split into two parts based on prediction point. The first part of positive window contains all events before prediction point, and we only take the first part of positive windows for re-ordering. On the other hand, for those windows which are labeled as negative, they stay unchanged without re-ordering because there is no private pattern based on speculation result, and we can proceed with the next coming window. Additionally, for positive windows, which would later enter into re-ordering phase, there should be at least two events that are part of private pattern in first part of positive window because only one event could not be re-ordered. If there is just one event in the first part of positive window, we shift the prediction point backwards until we observe at least two events in the first part of positive window.

4.5 **Top-k Preserving**

When dealing with events in re-ordering process, it is lack of consideration that only events belonging to private pattern are taken into consideration for re-ordering. If we select events that are part of public patterns for re-ordering, QoS will degenerate after re-ordering. In order to maintain high QoS, public patterns should also be taken into consideration before re-ordering. This is done by the top-k preserving approach. Those public patterns with higher weight are not selected for re-ordering. In this way, the top-k public patterns with highest weights are preserved, so high QoS is guaranteed after re-ordering.

As shown in Figure 4-6, the procedure of top-k preserving strategy is described. First of all, the input of top-k preserving phase is the first part of the positive window containing events, which are before prediction point. Before selecting types of preserved public patterns, all public patterns are divided into two groups: matched patterns and unmatched patterns for that window. The former represent the set of public patterns, which are detected in the incomplete input window, while the latter denote the set of public patterns, which do not exist in the window yet. Since matched patterns have already existed in input window, and they would definitely contribute towards QoS, they are all taken into consideration for re-ordering as “to be preserved”. On the other hand, there are two types for unmatched patterns: partly unmatched and totally unmatched patterns. We only consider partly unmatched patterns for preservation because if they are partly matched in the first part of positive window, the chance they occur in the remaining part is higher than those totally unmatched patterns. Therefore, partly unmatched patterns cannot be entirely removed since they will also have the contribution to QoS if they occur in the window. For simplicity, unmatched patterns are denoted as partly unmatched patterns in top-k preserving.

For example, private patterns, matched public patterns, unmatched public patterns are defined as below:

Private pattern = {‘A’, ‘B’, ‘C’}
Matched public patterns = {‘A’, ‘B’, ‘D’} ; weight = {6}
Unmatched public patterns = {‘B’, ‘C’, ‘D’}, {‘B’, ‘C’, ‘H’}, {‘B’, ‘C’, ‘I’}, {‘B’, ‘C’, ‘K’}, {‘B’, ‘C’, ‘L’} ; weight = {4}, {5}, {2}, {1}, {1}

In order to hide the private pattern, the pair for re-ordering can be selected based on combination of pairs derived from private pattern. In this case, we can re-order either pair {‘A’, ‘B’} or {‘B’, ‘C’}. If we only take matched public patterns into consideration, {‘A’, ‘B’, ‘D’} is the only preserving pattern, so pair {‘A’, ‘B’} cannot be re-ordered. Hence, {‘B’, ‘C’} is always selected for re-ordering. After re-ordering, the total weight is six units when only preserving matched public pattern {‘A’, ‘B’, ‘D’}. However, for instance, if unmatched public patterns {‘B’, ‘C’, ‘D’} and {‘B’, ‘C’, ‘H’} occur in the remaining part of positive window, the total weight after re-ordering is nine and it is lost when re-ordering pair {‘B’, ‘C’}. In other words, QoS may become lower without consideration of unmatched public patterns when they occur in the remaining part of positive window. Thus, unmatched public patterns should also be considered for preservation, and the top-k preserving strategy also includes unmatched public patterns in order to achieve higher QoS after re-ordering but with slightly lower weights when compared to the matched public patterns.

The main concept of top-k preserving strategy is to preserve *k* high-weight patterns, which may have high contribution towards QoS at the end. Therefore, unmatched public

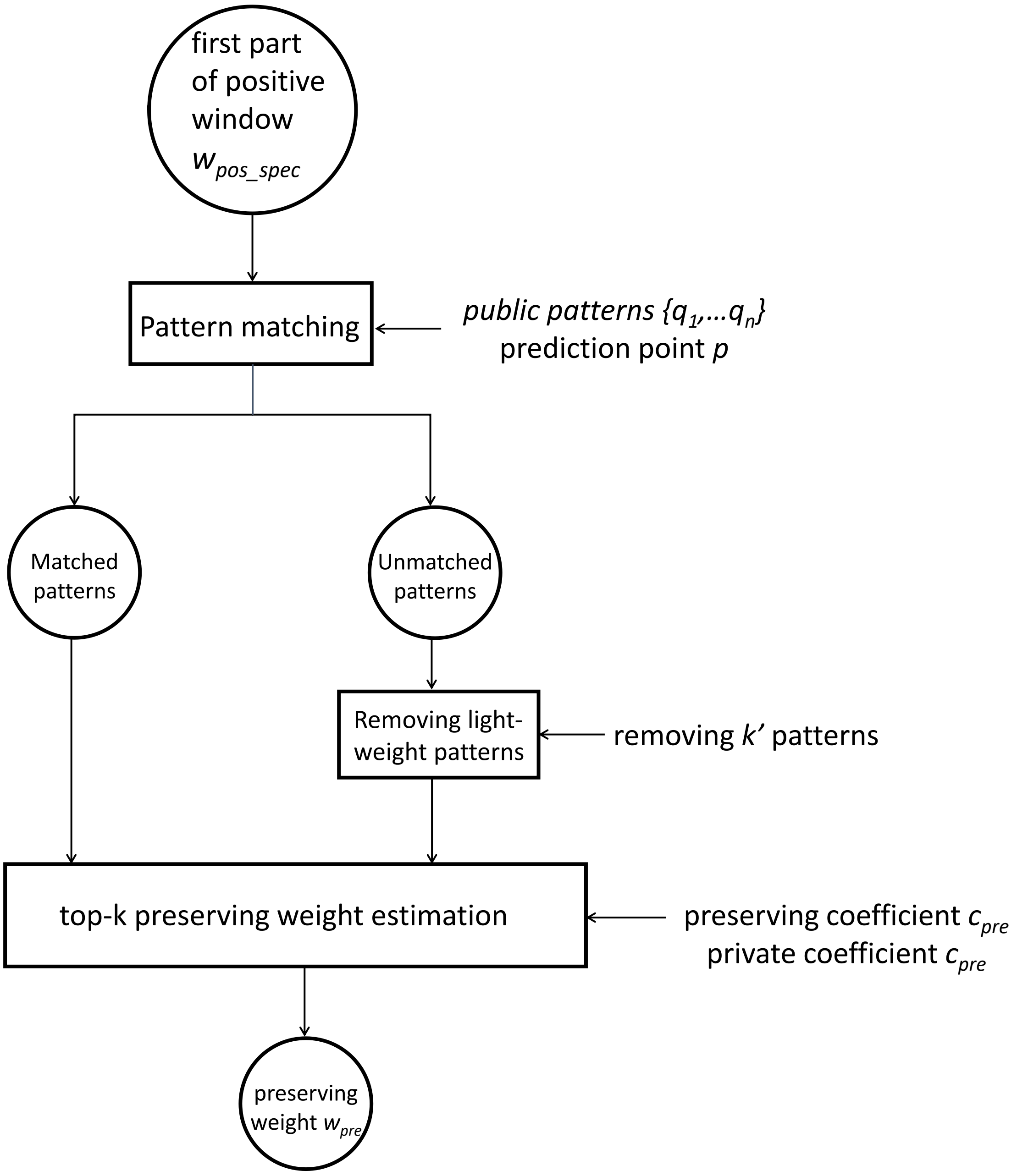

first part of positive window w_{pos_spec}
Pattern matching
public patterns $\{q_1,\ldots q_n\}$
prediction point p
Matched patterns
Unmatched patterns
Removing light-weight patterns
removing k' patterns
top-k preserving weight estimation
preserving coefficient c_{pre}
private coefficient c_{pre}
preserving weight w_{pre}

Figure 4-6. Procedure of top-k preserving

patterns with higher weight should firstly be preserved. Namely, we can remove the low-weight unmatched public patterns instead. To be more general, *k'* is defined as the removing number of unmatched patterns, and the number of preserving patterns is denoted as $k = N - k'$, where N is the total number of public patterns. In the previous example, assuming the removing number $k' = 2$, unmatched public patterns and their weights after applying top-k preserving strategy are shown as following:

Unmatched patterns after top-k preserving = {‘B’, ‘C’, ‘D’}, {‘B’, ‘C’, ‘H’}, {‘B’, ‘C’, ‘I’}

Weight of unmatched patterns after top-k preserving = {4}, {5}, {2}

In the above result, {‘B’, ‘C’, ‘K’} and {‘B’, ‘C’, ‘L’} are removed because they are the two lowest-weight unmatched patterns. As a consequence, all preserving patterns for re-ordering are determined as following:

Preserving patterns = {‘A’, ‘B’, ‘D’}, {‘B’, ‘C’, ‘D’}, {‘B’, ‘C’, ‘H’}, {‘B’, ‘C’, ‘I’}

Weights = {6}, {4}, {5}, {2}

However, it is difficult to determine the importance of every preserving pattern. The goal is to keep high QoS after re-ordering, so we can view this problem from QoS point of view. Since matched public patterns have already existed in input window, they would definitely have the contribution towards QoS. Thus, matched public patterns should be assigned to high proportion of weight. On the other hand, unmatched patterns have an influence on QoS only when they truly occur in the remaining part of positive window. Hence, lower proportion of weight is given to unmatched patterns. To this end, the preserving weight w_{pre} is introduced in top-k preserving weight estimation and w_{pre} is defined as following:

$$w_{pre} = \alpha \cdot \sum w_{matched} + \beta \cdot \sum w_{unmatched} \tag{4-15}$$

where $w_{matched}$ is the weight of matched public pattern for preserving; $w_{unmatched}$ is the weight of unmatched public pattern for preserving; α and β are the weighting factor for matched public patterns and unmatched public pattern, respectively.

As discussed before, matched patterns are assigned to higher proportion of weight, so α is greater than β. To simplify the equation (4-15), w_{pre} is rewritten as below:

$$w_{pre} = c_{pre} \cdot \sum w_{matched} + \sum w_{unmatched} \tag{4-16}$$

where c_{pre} is the preserving coefficient, which is defined as following:

$$c_{pre} = 1 + \frac{prediction\ point}{size\ of\ widow} \tag{4-17}$$

It is worth noting that since events before prediction point have already been observed before top-k preserving, we have to take these observed events into consideration when giving weights to matched public patterns. c_{pre} can be viewed as weighting factor for matched public patterns. Thus, in c_{pre}, the information of observed events are included by adding the ratio of prediction point to size of window. Moreover, if the prediction point is close to the back part of input window, there is more information about observed events, which implies that the input window contains more events in top-k preserving approach. As a result, c_{pre} is greater because more information of window is considered.

Accordingly, the pair for re-ordering can be determined if it has the greatest preserving weight based on top-k preserving strategy. The algorithm of top-k preserving is described in Algorithm 4.2. In the previous example, the re-ordered pair can be {'A', 'B'} or {'B', 'C'}. Assume that the prediction point is at the middle of processing window, and the preserving coefficient c_{pre} can be derived as below:

$$c_{pre} = 1 + \frac{prediction\ point}{size\ of\ widow} = 1 + 0.5 = 1.5 \tag{4-18}$$

When re-ordered pair is {'A', 'B'}, the preserving patterns are {'B', 'C', 'D'}, {'B', 'C', 'H'}, {'B', 'C', 'I'}, so the preserving weight $w_{pre,\{AB\}}$ can be derived with preserving factor c_{pre} as below:

$$\begin{aligned} w_{pre,\{AB\}} &= c_{pre} \cdot \sum w_{matched} + \sum w_{unmatched} \\ &= 1.5 \cdot 0 + (4 + 5 + 2) = 11 \end{aligned} \tag{4-19}$$

Similarly, when re-ordering pair {'B', 'C'}, the preserving pattern is only {'A', 'B', 'C'}, so the preserving weight is derived as following:

Algorithm 4.2 Top-k Preserving

```
procedure TOP-KPRESERVING(k’)
    $c_{pre}$ ← 1 + (predictionPoint / windowSize)
    preWeight ← {}
    if unmatchedPatternSize > k’ then
        for i ← 1 ... $k'$ do
            unmatchedPatterns.remove(lowest-weightPattern)
        end for
    else
        unmatchedPatterns.removeAll()
    end if
    for each pair in privatePattenPairSet do
        for pattern in unmatchedPatterns do
            if pattern is totallyUnmatchedPattern then
                unmatchedPatterns.remove(pattern)
                continue
            if pattern contains pair then
                unmatchedPatterns.remove(pattern)
            end if
        end for
        for pattern in matchedPatterns do
            if pattern contains pair then
                matchedPatterns.remove(pattern)
            end if
        end for
        weight ← $c_{pre} *$ sum(matchedPattern) + sum(unmatchedPattern)
        preWeight.add(weight)
    end for
    return preWeight
end procedure
```

$$w_{pre,\{BC\}} = 1.5 \cdot 6 + (0) = 9 \tag{4-20}$$

From equation (4-19) and (4-20), $w_{pre,\{AB\}}$ is greater than $w_{pre,\{BC\}}$, so the pair {BC} is selected for re-ordering after implementing top-k preserving strategy. In short, the top-k preserving strategy takes both matched and unmatched patterns into consideration, and the preserving weight is determined with the help of preserving coefficient c_{pre}.

4.6 Reordering

In re-ordering phase, the aim is to adjust the order of events that are part of re-ordered pair p_{re} with preserving weight derived from top-k preserving phase, and the input is the first part of positive window w_{pos_spec} containing events before prediction point p. The pattern-based access control mechanism is introduced in [11], and the graph-based re-ordering algorithm is implemented in this thesis. The overall flow of re-ordering phase is pictured in Figure 4-7.

First of all, a weighted directed acyclic graph is formed based on types of private patterns P_{priv} and public patterns $Q_1 \ldots Q_n$. In weighted directed graph, vertices represent events, and each directed edge represents the order of events. The weights of edge is based on timestamps of each event. After the graph is completed, the event order of re-ordered pair p_{pre} must be reversed. Thus, for event pairs with reversed order, the initial edge weights are negative because the timestamps in graph have not been modified according to the reversed order.

In graph-based re-ordering, the algorithm takes the formed graph as input. After detecting negative edges, the algorithm examines all vertices with weights representing event timestamps, and edge weights denoted as inter-arrival times. The examining iteration is completed until there is no negative edge. Therefore, all inter-arrival times are positive, which implies that timestamps are consistent with the event orders after re-ordering. The detailed algorithm about graph-based re-ordering is mentioned in [11].

Note that the processing window labeled as positive is divided into two parts (w_{pos_spec} and $w_{pos\_nonSpec}$) based on prediction point in speculation phase, and we take the first part of positive window w_{pos_spec} as an input for top-k preserving phase and re-ordering phase.

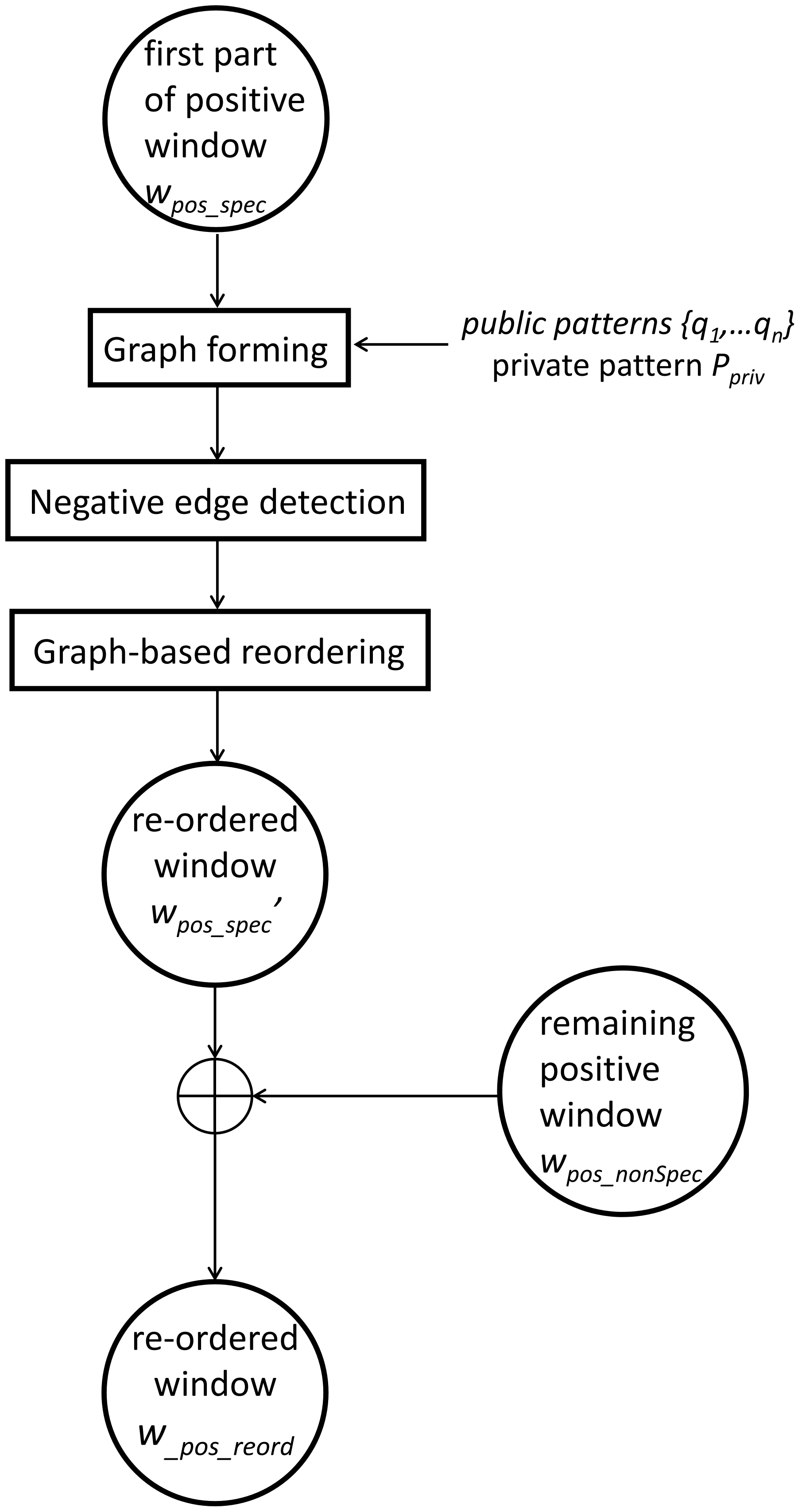

first part of positive window w_{pos_spec}
Graph forming
public patterns $\{q_1,\ldots q_n\}$
private pattern P_{priv}
Negative edge detection
Graph-based reordering
re-ordered window w_{pos_spec}'
remaining positive window $w_{pos\_nonSpec}$
re-ordered window $w_{\_pos\_reord}$

Figure 4-7. Flow of re-ordering

After re-ordering, the re-ordered window w_{pos_spec}' is obtained. By combining w_{pos_spec}' and $w_{pos\_nonSpec}$, the re-ordered positive window $w_{\_pos\_reord}$ is finally obtained.

4.7 Utility Comparator

At the end, we need to evaluate the proposed speculative re-ordering model. To be more precise about the impact of re-ordering, the utility is defined in [11] as below:

$$\begin{aligned}\text{Utility (U)} = {} & \Sigma_{i=1}^{\#\ of\ matched\ public\ patterns} w_i \\ & - 2 * \Sigma_{j=1}^{\#\ of\ matched\ false\ positives} w_j \\ & - \Sigma_{k=1}^{\#\ of\ matched\ private\ patterns} w_k; \qquad \text{(4-21)}\end{aligned}$$

where w_i, w_j are the user-defined weight of public pattern onto the QoS, and w_k is the weight of private patterns.

As shown in Figure 4-8, by comparing the original positive window w_{pos} and re-ordered positive window w_{pos}', false positives and false negatives can be observed. With false positives, false negatives, and matched public patterns, the utility is derived. It is worth noting that the high utility indicates there are more public patterns but less false positives, false negatives or private patterns after re-ordering. In other words, if there still exists a private pattern after re-ordering, the utility will become low since a private pattern cause a very high penalty.

Also, the utility of processing window based on non-speculative re-ordering algorithm in [11] is derived. In utility comparator, both utility are compared in terms of utility comparison ratio, which is defined as the ratio of utility of speculative-reordering to utility of non-speculative reordering. Note that the former should approach latter as closely as possible. Therefore, the aim is to achieve a high utility comparison ratio close to 1, and it would be used to evaluate our speculative re-ordering model.

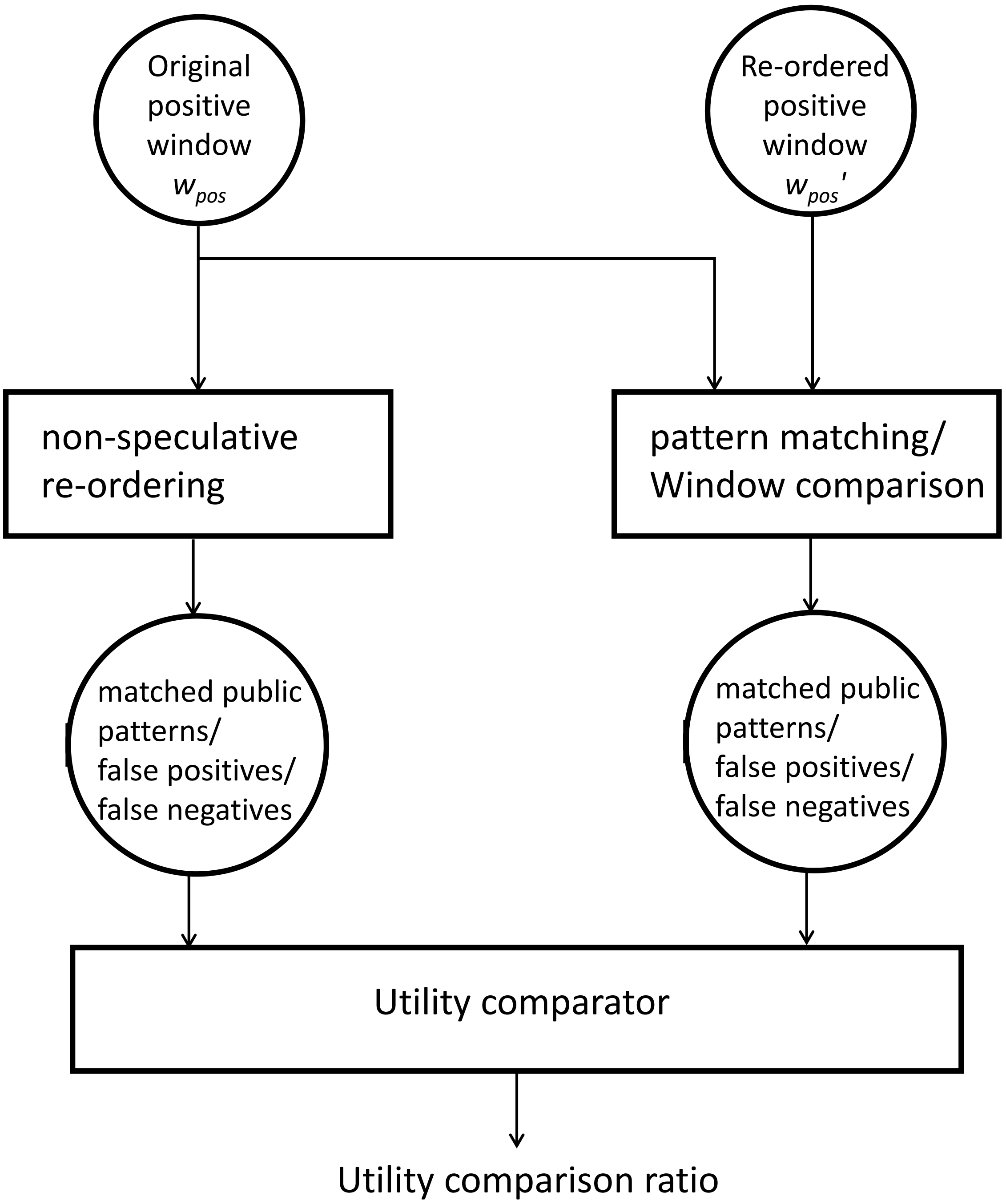

Original positive window w_{pos}
Re-ordered positive window w_{pos}'
non-speculative re-ordering
pattern matching/ Window comparison
matched public patterns/ false positives/ false negatives
matched public patterns/ false positives/ false negatives
Utility comparator
Utility comparison ratio

Figure 4-8. Flow of utility comparator

Chapter 5 Evaluation

In this chapter, we present evaluation results for the speculative Re-ordering strategy in terms of QoS and latency. Before evaluation, the setup of experiments is introduced. There are several factors that have an influence on the speculative re-ordering strategy. We evaluate the strategy with the following parameters.

- Precision and recall: The first and foremost evaluation is the accuracy of speculation. We use precision and recall, the well-known model for evaluation the accuracy or performance of speculative model based on Markov chains.
- Threshold t_s: Threshold t_s determines the confidence probability with which the private pattern match is predicted.
- Prediction point *p*: The prediction point *p* represents the point of start of prediction in terms of percentage of window size.
- Window size: The window size determines the total number of events in which need to be considered for speculation and re-ordering.
- Removing number *k'*: Instead of preserving the most important *k* public patterns, we can remove *k'* public patterns with lowest weight.
- Running time: The running time impacts whether we can save the latency when applying speculative re-ordering strategy.

5.1 Setup and Parameters

We implement both non-speculative and speculative re-ordering strategy based on graph-based re-ordering mechanism in [11] for our evaluation. Both the systems take the input data stream, and return the modified data stream as output. For all experiments, the programming language is Python 2.7. The experimental machine used is an Intel Core i5-6200U 2.8GHz CPU and the operating system is Windows 10 and the RAM is 8GB.

The input data is randomly generated with 25000 windows, and the characteristic of input data is assumed to be same as the training dataset. We assume that windows in every experiment are all event-based, and the event types are composed of all English alphabets in lower and upper case for the purpose of generating public patterns and the private pattern.

5.2 Utility Comparison

With the definition of utility discussed in equation (4-21), the utility comparison ratio can be derived as the ratio of speculative reordering to non-speculative reordering algorithm. Thus, the impact on QoS is evaluated for speculative re-ordering strategy in comparison to non-speculative re-ordering strategy with the utility comparison ratio. For all following evaluations, we use this ratio to measure utility performance.

Furthermore, latency is evaluated by measuring running time of speculation, top-k preserving and re-ordering. For speculative re-ordering strategy, since it often takes very short time for speculation, the running time for speculation and top-k preserving are combined. After top-k preserving, the re-ordering time is measured. On the other hand, there is only re-ordering time for non-speculative re-ordering strategy. Note that we use the resolution of microseconds for time evaluation, so if the measuring time is less than one micro second, it will be viewed as zero.

5.3 Evaluation of Proposed Speculative Model

The proposed speculative model is evaluated by precision and recall in test phase. The test dataset is generated by random selection from event types. The characteristic of test dataset is basically same as training dataset. Additionally, there are 25000 windows in test dataset, and each window contains 100 events. The setting of parameters are given by following table:

Parameters	Private pattern	Threshold range	Increment
Value	{‘A’, ‘B’, ‘C’, ‘D’}	0.55 – 0.925	0.025

Table 5-1 Setting of evaluation of proposed speculative model

First of all, the input stream is taken into the speculative model based on Markov chains. With given thresholds, the corresponding prediction points are obtained. Also, every incoming window is labeled as positive or negative based on speculation results. At the end, recall and precision are calculated.

The corresponding prediction points are shown in Figure 5-1 for changing thresholds.

It is worth noting that all prediction points are located before 50% of window is completed. The results of precision and recall are shown in Figure 5-2 with different given thresholds.

In Figure 5-2, it can be viewed as two zones split by the threshold 0.675. In the front zone, the average precision is around 0.8, and the maximum recall is nearly 0.6. In the back zone, the average precision is around 0.95 but the maximum recall is less than 0.3. Hence, an optimal threshold should be selected depending on the requirements. We select the optimal threshold of 0.55 such that both precision and recall is the best. The corresponding point prediction point is 46% of window size. In this case, the precision is 0.75 and the recall is 0.6. Thus, for all windows labeled as positive, only 1/4 is wrong, and there are nearly 40 % missing windows which contains private pattern.

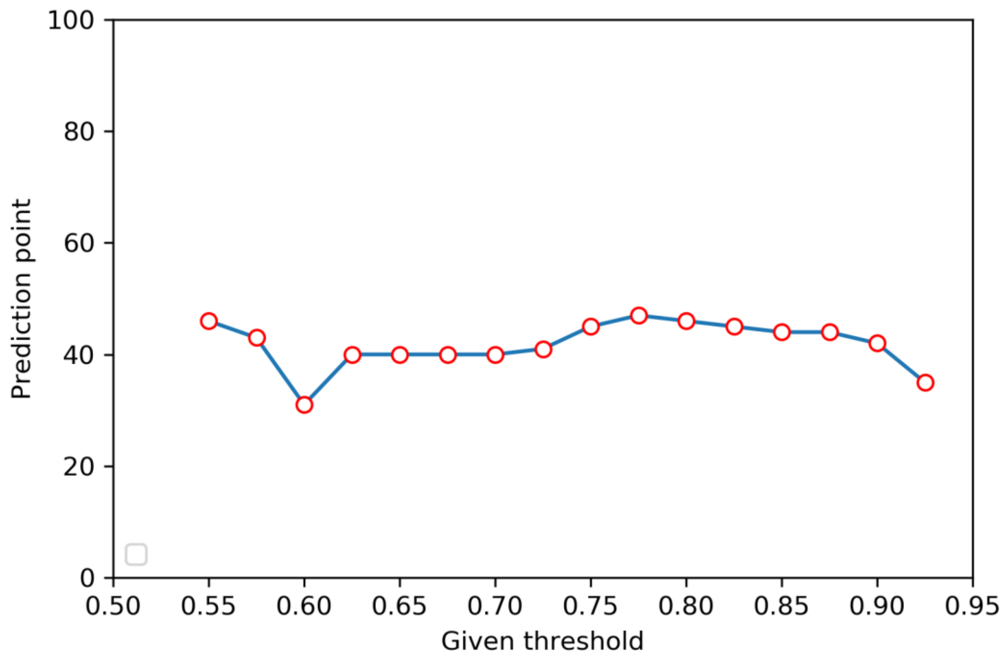


Figure 5-1 Prediction points by given thresholds

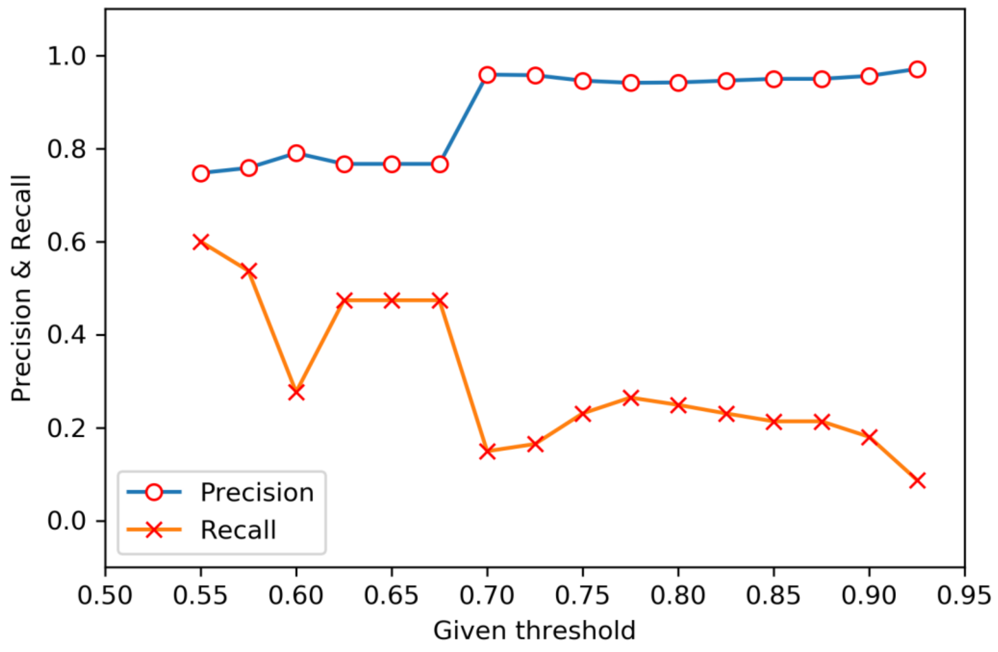


Figure 5-2 Precision and recall by given thresholds

5.4 Evaluation of Threshold

The threshold is the parameter to determine whether the private pattern occurs in current window when the transition probability is greater than threshold. In this section, this evaluation can also be used as an alternative to offline learning, where they can find the optimal threshold based on precision and recall. Therefore, in order to obtain the optimal parameters for high utility comparison ratio, we start speculation with different given thresholds derived from previous offline learning. In this experiment, the setting of parameters are given by following tables:

Parameters	Window size	Private pattern	Number of public patterns
Value	100	{‘A’, ‘B’, ‘C’, ‘D’}	15

Table 5-2 Setting (I) of evaluation of threshold

Parameters	Removing number k’	Threshold range	Increment
Value	2	0.55 – 0.925	0.25

Table 5-3 Setting (II) of evaluation of threshold

The result of utility comparison ratio is shown in Figure 5-3 by given thresholds. For all utility comparison ratios based on different thresholds in Figure 5-3, they are all in the range from 70% to 76%. Based on this result, we select the optimal threshold as 0.775 for speculation.

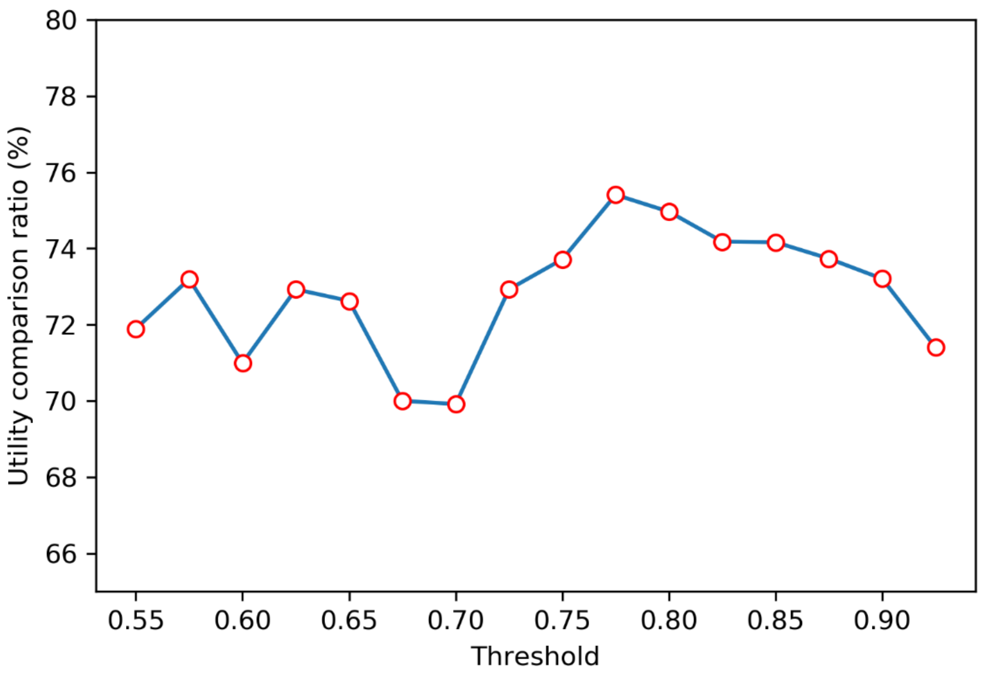


Figure 5-3 Utility comparison ratio under evaluation of threshold

5.5 Evaluation of Prediction Point

The prediction point *p* determines how many events should be observed before the state of speculation and re-ordering. With the optimal threshold derived in previous experiment, we would like to evaluate at which prediction point we can obtain the high utility comparison ratio after speculation. In this experiment, we first start speculation with the optimal threshold of 0.775, and every time we shift prediction point with 10% of window size until the prediction point is over 90% of window size. The setting of parameters are given by following tables:

Parameters	Window size	threshold	Private pattern	Number of public patterns
Value	100	0.775	{‘A’, ‘B’, ‘C’, ‘D’}	15

Table 5-4 Setting (I) of evaluation of prediction point

Parameters	Removing number k’	Initial prediction Point	Increment
Value	2	48	10

Table 5-5 Setting (II) of evaluation of prediction point

The result of utility comparison ratio is shown in Figure 4 by given prediction points. As shown in Figure 5-4, the utility comparison ratio increases when the prediction point is close to the end of window. When prediction point is closer to the end of window, there are more events observed before speculation. This is because we have more information about appeared events and event orders, so the speculation would be more accurate. As a consequence, the utility comparison ratio increases when the prediction point increases.

5.6 Evaluation of Window Size

In this section, we evaluate the speculation strategy with window size. We assume that the number of windows is always fixed to 25000. For different window sizes, the number of events in window is different, so the total number of events in test dataset is also different. In this experiment, we generate the test dataset for different size of windows, and obtain corresponding thresholds and prediction points by using offline

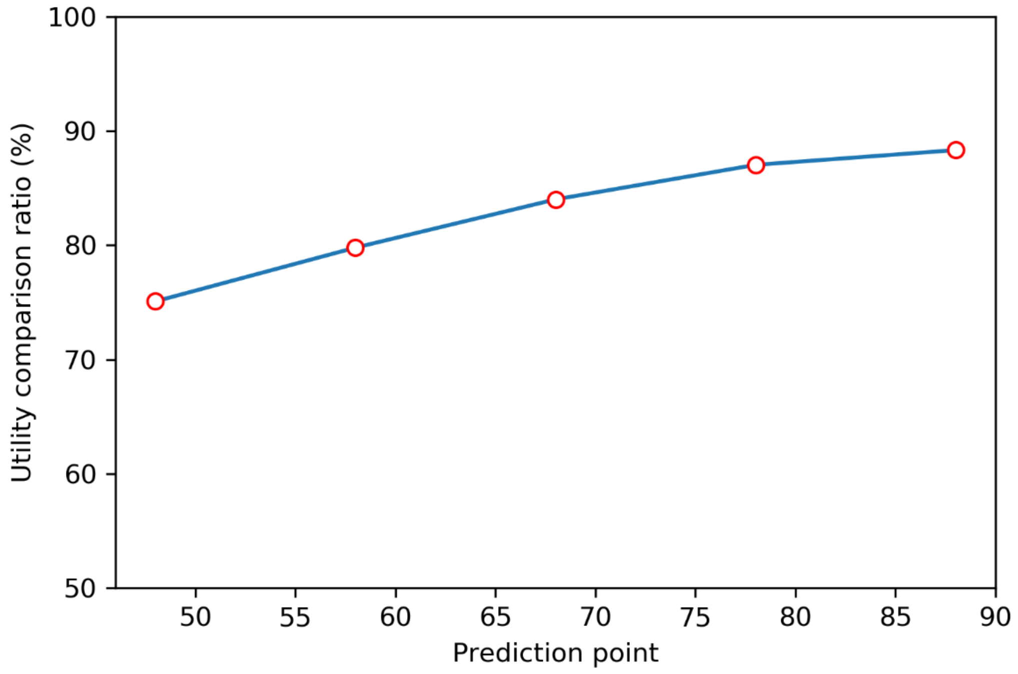


Figure 5-4   Utility comparison ration under setting of prediction point

learning. With same public patterns, we then start speculative re-ordering strategy with proposed model in order to acquire utility comparison ratio for different window sizes. The setting of parameters are given by the following tables:

Parameters	Private pattern	Number of public patterns
Value	{‘A’, ‘B’, ‘C’, ‘D’}	15

Table 5-6 Setting (I) of evaluation of window size

Parameters	Removing number k’	Range of window size	Increment
Value	4	100 - 200	20

Table 5-7 Setting (II) of evaluation of window size

As shown in Figure 5-5, the utility comparison ratio increases when the window size becomes bigger. If the size of window increases, there are more events in each window, and it is more likely that more public patterns occur. During re-ordering, only public patterns that overlapped with the private patterns will be affected, while other public patterns are kept unchanged. Thus, there may exist more unchanged public patterns in a bigger window, which results in higher utility comparison ratio. The result also shows that our strategy works well for all windows.

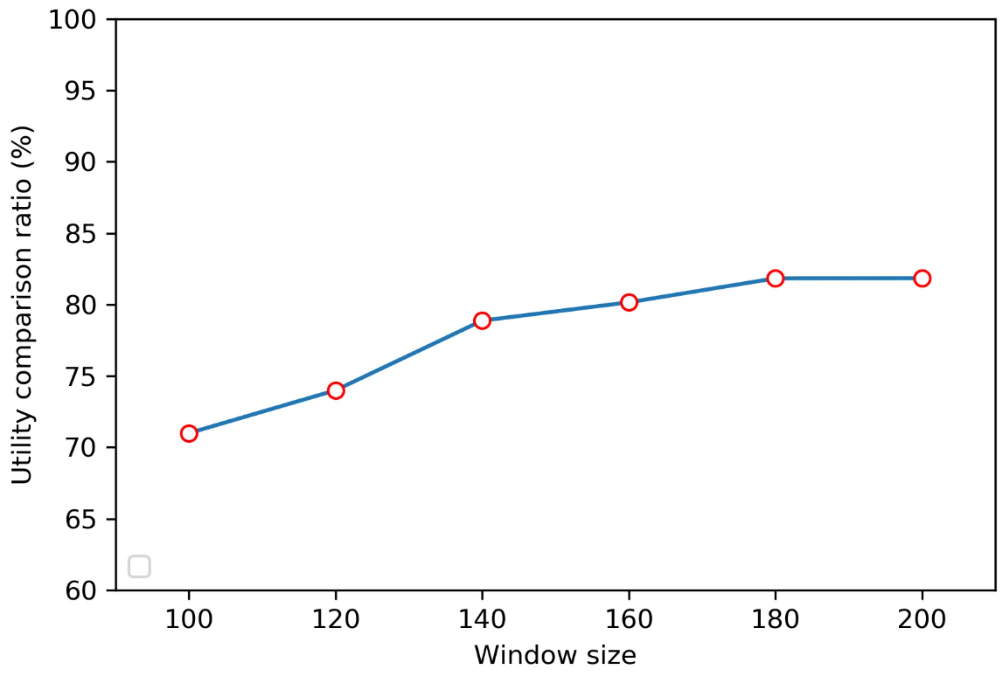


Figure 5-5 Utility comparison ratio under the setting of window size

5.7 Evaluation of Removing Number ***k'***

The removing number *k'* determines how many public patterns we remove in the top-k preserving strategy. In this experiment, we start speculative re-ordering strategy by increasing *k'*, and at the end compare the results with and without preserving public patterns. The evaluation without preserving public patterns is to show the importance of top-k preserving strategy. The setting of parameters are given by following tables:

Parameters	Window size	threshold	Prediction point	Precision	Recall
Value	100	0.55	46	0.75	0.6

Table 5-8 Setting (I) of evaluation of removing number *k'*

Parameters	Private pattern	Number of public patterns	Range of removing number k'
Value	{'A', 'B', 'C'}	15	0 – 4, all

Table 5-9 Setting (II) of evaluation of removing number *k'*

In Figure 5-7, results of utility comparison ratio are shown with different *k'*. For all *k'*, their utility comparison ratios are nearly 80%, and there is no significant difference among them. Since the weights of our public patterns are all less than 10, which is quite smaller than total utility, the preserved public patterns show little contribution towards total utility.

However, instead of utility comparison ratio, the result in Figure 5-8 shows a considerable difference in terms of number of false negatives. When preserving no public pattern, the number of false negatives is around 2600, which implies that the number of missing public pattern is almost 2600 after reordering. On the other hand, the average number of false negatives is less than 250 when taking top-k preserving algorithm into consideration with removing number from 0 to 4. From this result, it provides us a strong evidence that top-k preserving strategy is useful in terms of number of false negatives. Furthermore, in Figure 5-8, the number of false negatives are compared when preserving public patterns. When all public patterns are preserved, the number of false negatives is almost half of the number of false negatives in other cases. This is because of the obvious reason that the re-ordering strategy tries to preserve the given of public patterns until concealing private patterns.

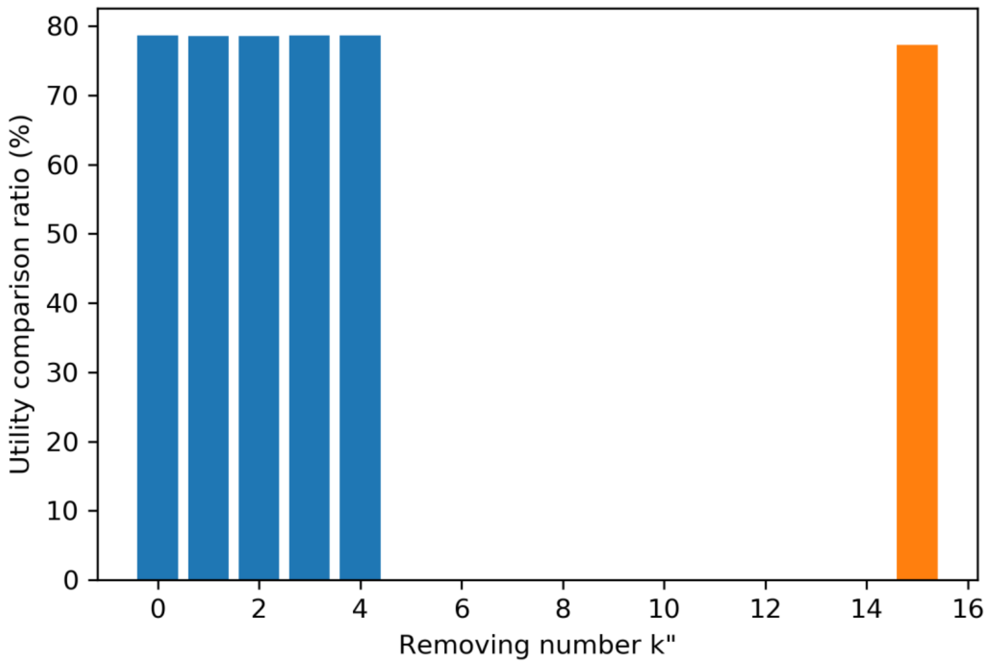


Figure 5-6 Utility comparison ratio with different removing numbers

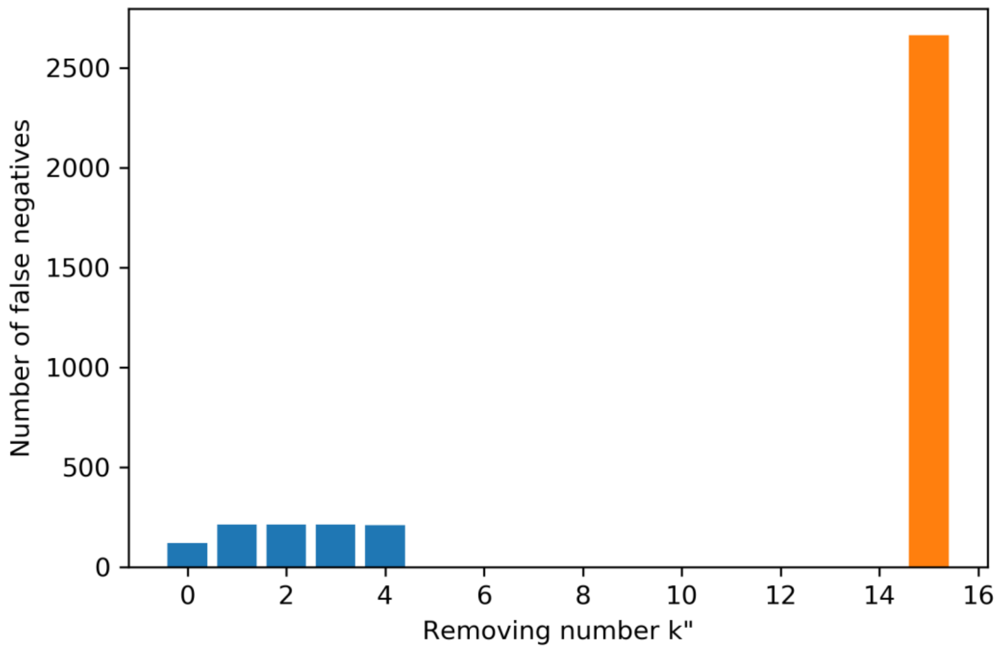


Figure 5-7 Number of false negatives with different removing numbers

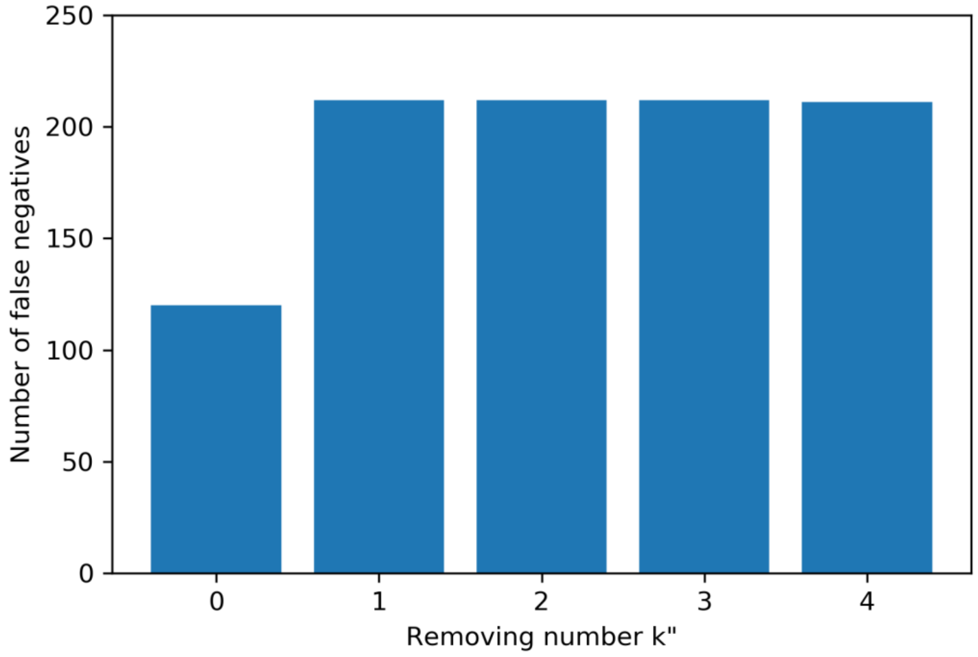


Figure 5-8 Number of false negatives with removing number form 0 to 4

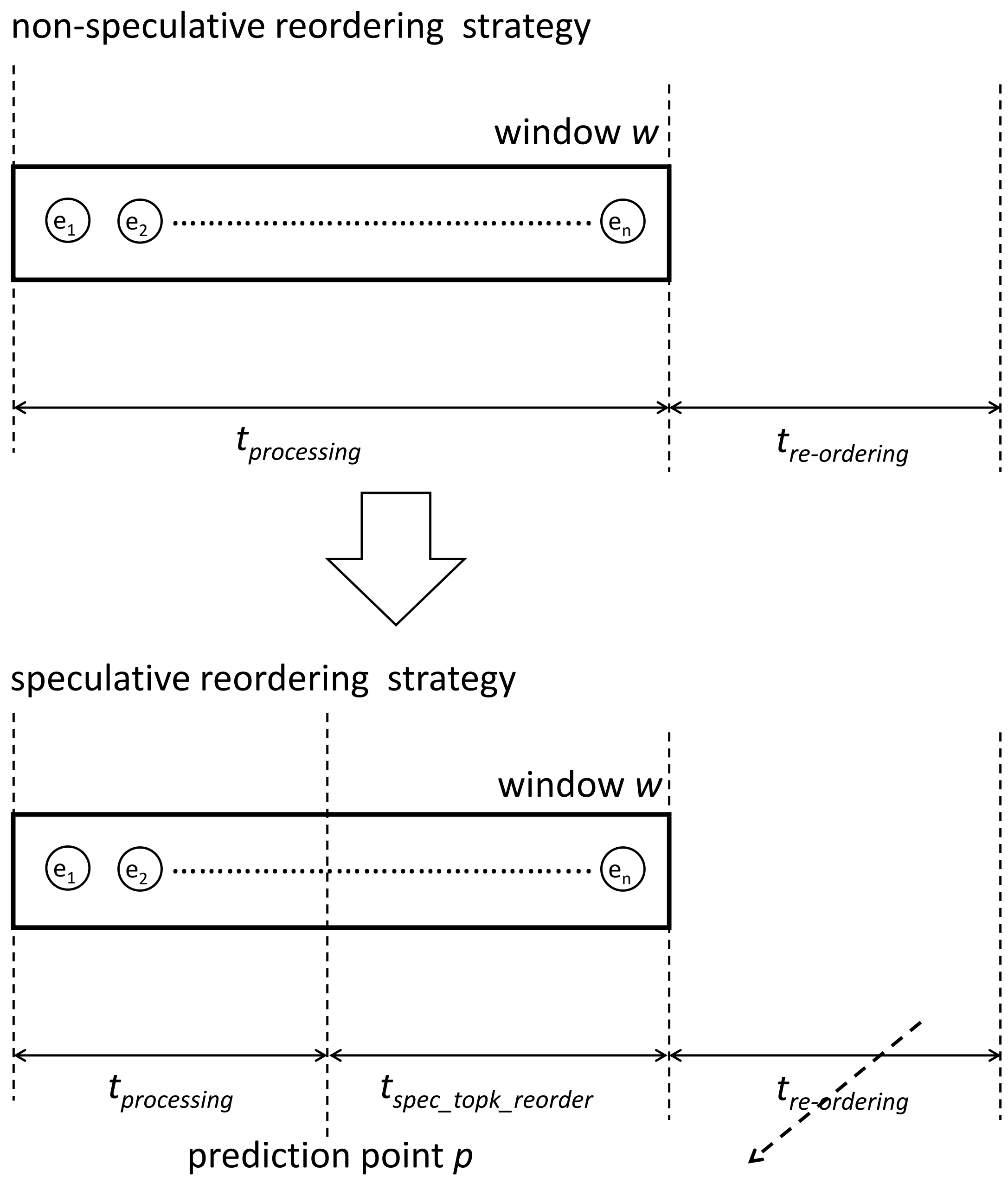


Figure 5-9 Comparison of speculative and non-speculative reordering strategy

5.8 Evaluation of Time

In order to evaluate how much latency can be saved while implementing speculative re-ordering strategy, we measure the evaluation time for speculation, top-k preserving and re-ordering. As shown in Figure 5-9, for non-speculative strategy, the re-ordering always starts from the end of window after observing all events. Therefore, it takes re-ordering time $t_{re\text{-}ordering}$ to conceal the private pattern while implementing non-speculative re-ordering strategy. The total time will increase if the size of window becomes bigger. On the other hand, when applying speculative re-ordering strategy, there is no need to wait until the end of window. It starts speculation after prediction point p, and $t_{spec\_topk\_reorder}$ is denoted as total time of speculation, top-k preserving and re-ordering. If $t_{spec\_topk\_reorder}$ is completed before the end of window, we can save the re-ordering time $t_{re\text{-}ordering}$ of non-speculative strategy. In other words, the latency is improved by implementing speculative re-ordering strategy. Therefore, for speculative re-ordering strategy, $t_{spec\_topk\_reorder}$ should not be less than remaining time after prediction point, which can be described as below:

$$t_{spec\_topk\_reorder} \leq T \cdot \frac{window\ size - p}{window\ size} \qquad (5\text{-}1)$$

where T is the processing time for each window.

Thus, T in equation (5-1) can become as below:

$$T \geq \frac{window\ size}{window\ size - p} \cdot t_{spec\_topk\_reorder} \qquad (5\text{-}2)$$

Furthermore, in this experiment, the setting of parameters are given by following tables:

Parameters	Window size	threshold	Prediction point	Precision	Recall
Value	100	0.55	46	0.75	0.6

Table 5-10 Setting (I) of evaluation time

Parameters	Private pattern	Number of public patterns
Value	{‘A’, ‘B’, ‘C’, ‘D’}	15

Table 5-11 Setting (II) of evaluation time

The results of evaluation time are shown as in Table 5-12 for both non-speculative and speculative re-ordering strategy with *k'* from 0 to 4.

Removing number *k'* / Average evaluation time (us)	*k'* = 0	*k'* = 1	*k'* = 2	*k'* = 3	*k'* = 4
Speculation/Top-k preserving	110.32	101.52	103.36	116.12	96.96
Re-ordering	23373.32	1515.92	1479.32	1538.96	1449.68
Total time $t_{spec\_topk\_reorder}$	23483.64	1617.44	1582.68	1655.08	1546.64

Table 5-12 Average evaluation time of speculative re-ordering strategy among 25000 windows

Average evaluation time (ms)	Non-speculative re-ordering strategy
Re-ordering	355.481

Table 5-13 Average evaluation time of non-speculative re-ordering strategy among 25000 windows

Based on the result of evaluation time in Table 5-12, when preserving all public patterns, it would take longer time for re-ordering while the re-ordering time is shorter when removing some low-weight public patterns. As a consequence, there are mainly two situations in order to save the latency compared to non-speculative re-ordering strategy. First, when our users take the number of false negatives into main consideration, we should preserve all public patterns (k' = 0) because the number of false negatives is lowest based on result of previous evaluation. In this way, with p = 46, window size = 100 and the result of evaluation time when removing number k' is 0, equation (5-2) can be derived as below:

$$T \geq 43.49 \text{ ms} \tag{5-3}$$

Second, if the number of false negatives is not the main concern for our users, since the result of previous evaluation shows that there is no substantial difference of utility comparison rate among different removing numbers, we can choose removing number *k'* as 4 for our users, and it gives the shortest $t_{spec\_topk\_reorder}$. Similarly, equation (5-2) can be also derived as below:

$$T \geq 2.86 \text{ ms} \tag{5-4}$$

From equation (5-3) and equation (5-4), we can conclude that the window size should be at least 43.49 ms when our users care more about the number of false positive. Also, the processing time of window should be at least 2.86 ms when the requirements are more stringent for latency. In general, these minimum window sizes are sufficient for a very large numbers of CEP applications. In other words, our strategy eliminates the need for re-ordering time if the window size is greater than 50 ms. Furthermore, in both situations, we can save 355.481 ms on an average in our examples, which is the additional latency because of re-ordering when implementing non-speculative re-ordering strategy.

Chapter 6 Conclusion

In this thesis, we propose the model with speculative re-ordering strategy in order to maintain pattern-based privacy in CEP systems. The current available mechanism based on re-ordering algorithm takes the long latency when the size of window increases, so we extend its application by speculating whether the private pattern occurs before re-ordering. For public patterns, the top-k preserving algorithm is introduced in order to preserve relating important public patterns during re-ordering, thus decreasing the number of false negatives after re-ordering.

In order to evaluate the impact of QoS, utility comparison ratio is defined as the performance metric. The evaluation result shows that the utility comparison ratio is nearly 80% for the test dataset generated in our evaluation. When implementing top-k preserving algorithm for public patterns, the number of false negatives is decreased by 90 % in comparison to preserving no public pattern. Also, the latency is evaluated by measuring the running time for speculation, top-k preserving and re-ordering when implementing both speculative and non-speculative re-ordering strategy. We can conclude that for our test dataset, we can eliminate the time taken for re-ordering completely if the window size is greater than 3 ms.

REFERENCE

[1] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman. Efficient pattern matching over event streams. In SIGMOD, 2008.

[2] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In SIGMOD, 2006.

[3] Y. Mei and S. Madden. Zstream: A cost-based query processor for adaptively detecting composite events. In SIGMOD, 2009.W. T.

[4] Microsoft Stream Insight: http://msdn.microsoft.com/en-us/sqlserver/ee476990.aspx.

[5] StreamBase: www.streambase.com.

[6] Raman Adaikkalavan, Indrakshi Ray, and Xing Xie. 2011. Multilevel Secure Data Stream Processing. In Data and Applications Security and Privacy XXV, Yingjiu Li (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 122–137.

[7] Matteo Migliavacca, Ioannis Papagiannis, David M. Eyers, Brian Shand, Jean Bacon, and Peter Pietzuch. 2010. DEFCON: High-performance Event Processing with Information Security. In Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference (USENIXATC’10). USENIX Association, Berkeley, CA, USA, 1–1.

[8] Jianneng Cao, Barbara Carminati, Elena Ferrari, and Kian-Lee Tan. 2009. ACStream: Enforcing Access Control over Data Streams. In 2009 IEEE 25th International Conference on Data Engineering. IEEE.

[9] G. Cugola, A. Margara, "Processing flows of information: From data stream to complex event processing", ACM Computing Surveys (CSUR), vol. 44, no. 3, pp. 15, 2012.

[10] Di Wang, Yeye He, Elke Rundensteiner, and Jeffrey F. Naughton. 2013. Utility maximizing event stream suppression. In Proceedings of the 2013 international conference on Management of data - SIGMOD '13. ACM Press.

[11] Saravana Murthy Palanisamy, Frank Dürr, Muhammad Adnan Tariq, Kurt Rothermel. 2018. Preserving Privacy and Quality of Service in Complex Event Processing through Event Reordering. In DEBS '18 Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems.

[12] R. Vilalta and Sheng Ma. 2002. Predicting rare events in temporal domains. In

ICDM.

[13] Carlotta Domeniconi, Chang-shing Perng, Ricardo Vilalta, and Sheng Ma. 2002. A Classication Approach for Prediction of Target Events in Temporal Sequences. In Principles of Data Mining and Knowledge Discovery. Springer.

[14] Gary M. Weiss and Haym Hirsh. 1998. Learning to Predict Rare Events in Event Sequences. In KDD.

[15] R. Vilalta and Sheng Ma. 2002. Predicting rare events in temporal domains. In ICDM.

[16] Asela Gunawardana, Christopher Meek, and Puyang Xu. 2011. A Model for Temporal Dependencies in Event Streams. In Advances in Neural Information Processing Systems 24. Curran Associates, Inc.

[17] Vinod Muthusamy, Haifeng Liu, and Hans-Arno Jacobsen. 2010. Predictive Publish/Subscribe Matching. In DEBS. ACM.

[18] Elias Alevizos, Alexander Artikis, and George Paliouras. Event forecasting with pattern markov chains. In Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS '17, pages 146–157. ACM, 2017.

[19] Ruben Mayer, Ahmad Slo, Muhammad Adnan Tariq, Kurt Rothermel, Manuel Gräber, and Umakishore Ramachandran. 2017. SPECTRE: Supporting Consumption Policies in Window-based Parallel Complex Event Processing. In Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference (Middleware '17). ACM, New York, NY, USA, 161–173. https://doi.org/10.1145/3135974.3135983

[20] Yu Wang, Gao Cong, Guojie Song, Kunqing Xie: Community-based greedy algorithm for mining top-k influential nodes in mobile social networks. In Proc. 16th Internat. Conf. on Knowledge Discovery and Data Mining (KDD'10), pp. 1039–1048, 2010. [doi:10.1145/1835804.1835935]

[21] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. ACM Comp. Surveys, 40(4):1–58, 2008.

[22] Yan Chen, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. 2017. PeGaSus: Data-Adaptive Differentially Private Stream Processing. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17). ACM, New York, NY, USA, 1375–1388.

[23] Do Le Quoc, Martin Beck, Pramod Bhatotia, Ruichuan Chen, Christof Fetzer, and Thorsten Strufe. 2017. Privacy Preserving Stream Analytics: The Marriage of Randomized Response and Approximate Computing. CoRR abs/1701.05403 (2017). arXiv:1701.05403

[24] A Design Framework for Internet-Scale Event Observation and Notification

[25] Processing Flows of Information: From Data Stream to Complex Event Processing

[26] Chet Geschickter and Kristin R. Moyer. 2016. Measuring the Strategic Value of the Internet of Things for Industries. Technical Report TR ID : G00298896. Gartner Inc

[27] Avesh Singh. 2017. Applying Artificial Intelligence in Medicine. (May 2017). Retrieved 2018-02-26 from https://blog.cardiogr.am/applying-artificialintelligence-in-medicine-our-early-results-78bfe7605d32

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature

Declaration